José Eduardo Marques Pimenta

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

Configurador de sistemas distribuídos de controlo especificados através de redes de Petri IOPT

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletrotécnica e de Computadores

Orientador : Luis Gomes, Professor Doutor, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri:

Presidente: Doutor Rui Manuel Leitão Santos Tavares - FCT/UNL

Vogais: Doutor Luis Filipe dos Santos Gomes - FCT/UNL (Orientador)
Doutora Anikó Katalin Horváth da Costa - FCT/UNL (Arguente)

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2015

Configurador de sistemas distribuídos de controlo especificados através de redes de Petri IOPT

Agradecimentos

Quero agradecer ao professor Luis Gomes por toda a paciência que teve já que esta tese não demorou pouco tempo a ser feita e também agradecer toda a orientação e disponibilidade que demonstrou que esta tese fosse terminada.

Um agradecimento a toda a minha família porque sem eles a realização da mesma não seria possível e também todo a força que me deram para que esta fosse terminada.

À minha mãe um especial agradecimento pela motivação que me deu para que nunca desistisse de terminar este trabalho e por ter estado sempre presente quando era necessário.

Ao meu irmão, pela disponibilidade que teve em me ajudar nas alturas mais difíceis e que me ajudou a continuar a desenvolver esta ferramenta.

À minha namorada, quero agradecer pela paciência que teve e também por compreender as vezes que não podemos estar juntos ou que não aproveitamos melhor as férias ou os fins de semana por ter de ficar a fazer a tese.

A todos os meus amigos de Faculdade pois sem eles não seria possível completar esta etapa e também para agradecer os bons momentos passados.

A todos os outros amigos também quero deixar uma palavra de agradecimento por algumas vezes vos ter falhado para que pudesse acabar este projeto.

As empresas em que trabalhei um agradecimento também especial por algumas trocas de turnos e por me deixarem sair as vezes um pouco mais cedo para que me pudesse reunir com o meu professor e assim ver mais perto a conclusão desta tese.

Os trabalhos associados a esta dissertação beneficiaram de resultados obtidos no projeto Petri-Rig - Ambiente de desenvolvimento de sistemas embutidos baseado em redes de Petri, financiado pela FCT com a referência PTDC/EEI-AUT/2641/2012.

Resumo

O objetivo principal deste trabalho é desenvolver um protótipo de ferramenta que permita a geração de ficheiros de configuração de sistemas distribuídos de controlo em plataformas específicas permitindo a integração de um conjunto de componentes previamente definidos.

Cada componente é caracterizado como um módulo, identificando-se o conjunto de sinais e eventos de entrada e saída, bem como o seu comportamento, normalmente especificado através de um modelo em redes de Petri IOPT – RdP-IOPT (*Input-Output Place-Transitions*). O formato PNML (*Petri Net Markup Language*) será utilizado para a representação de cada componente.

Os componentes referidos poderão ser obtidos através de vários métodos, nomeadamente através de ferramentas em desenvolvimento, que se encontram disponíveis em http://gres.uninova.pt/IOPT-Tools/ e também através da sua edição no editor de IOPT, como resultado da partição de um modelo expresso em IOPT, utilizando o editor Snoopy-IOPT em conjugação com a ferramenta SPLIT.

Serão considerados várias formas para interligação dos componentes, incluindo-se ligações diretas e *wrappers* assíncronos num contexto de sistemas Globalmente Assíncronos Localmente Síncronos - GALS bem como diferentes tipos de barramentos e ligações série, incluindo *Network-On-Chip* específicos.

A descrição da interligação entre componentes é gerada automaticamente pela ferramenta desenvolvida, tendo em conta resultados de dissertações de mestrado anteriores. As plataformas especificas de suporte à implementação incluem FPGA's da serie *Xilinx Spartan3*,3E e *Xilinx Virtex*, e várias placas de desenvolvimento.

Palavras chave: configurador, PNML, Network-On-Chip, wrappers, IOPT

Abstract

The main goal of this work is to develop a tool prototype that enables the generation of a configuration file for specific platforms allowing the integration of a predefined set of components.

Each component is characterized as a module, identifying the set of input and output signals and events, as well as their behavior, usually specified using IOPT (Input-Output Place-Transition) Petri nets. The PNML (Petri Net Markup Language) format will be used for the representation of each component.

The referred components could be obtained using several methods, namely through tools in development publicly available at http://gres.uninova.pt/IOPT-Tools/, and also through their edition in the IOPT editor, as a result of the partition of the IOPT model, using the Snoopy-IOPT editor in conjunction with the SPLIT tool.

Several ways are considered to interconnect the components, including direct connections and asynchronous wrappers in a context of Globally-Asynchronous Locally-Synchronous (GALS) systems, as well as different types of buses and serial links, including specific Network-On-Chip.

The description of the connections between components is automatically generated by the developed tool, taking into account results of already concluded MSc theses. The specific platforms explicitly supported include several FPGA's of series Xilinx Spartan3,3E and Xilinx Virtex and boards.

Keywords: Configurator, PNML, Network-on-Chip, wrappers, IOPT

Conteúdo

Lista de Figuras

Lista de Tabelas

Simbologia e notações

AC	Assynchronous Channel
CDMA	Code-Division Multiple Acess
CLBs	Configurable Logic Blocks
CCN	Central Coordination Node
CRC	Cyclic Redundancy Check
CSS	Cascading Style Sheet
FIFO	First In First Out
GALS	Globally-Asynchronous Locally-Synchronous
HTML	HyperText Markup Language
LSI	Locally Synchronous Island
NoC	Network-on-Chip
OSI	Open System Initiative
PNML	Petri Net Markup Language
RdP	Rede de Petri
RdP-IOPT	Rede de Petri Input-Output-Place-Transition
SDLC	Semi-Decoupled Latch Controller
SGML	Standard Generalized Markup Language
SoCs	System-on-Chip
SPIN	Scalable Programmable Integrated Network
TD	Time Domain
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XST	Xilinx Synthesis Tecnhology

1

Introdução

1.1 Enquadramento

A evolução dos circuitos digitais tem vindo a permitir a integração de múltiplos componentes num único chip, a evolução dos semicondutores permitiu uma diminuição do tamanho dos circuitos e permitiu aumentar o número de dispositivos num só integrado. Este tipo de solução pode utilizar dispositivos reconfiguráveis ou circuitos dedicados. Os sistemas descritos anteriormente são denominados *System-on-Chip* (SoCs).

Os sistemas síncronos continuam a dominar a maioria dos circuitos digitais, apesar do grande aumento de complexidade e velocidade dos circuitos integrados. O seu contínuo aumento tornará estes sistemas tão complexos que será improvável que um *clock* comum consiga ser distribuído por um chip de grande complexidade. Neste cenário, o modelo de *clock* síncrono centralizado não será desejável, forçando a uma alteração para abordagens assíncronas.

Uma das arquiteturas desenvolvidas para o funcionamento no modo assíncrono é a *Globally-Asynchronous Locally-Synchronous* (GALS). As GALS comunicam assincronamente entre os diferentes componentes síncronos. Os sistemas GALS combinam os benefícios dos sistemas síncronos e assíncronos, tornado-os assim numa arquitetura mais complexa, mas com mais benefícios. Outra arquitetura proposta é *Network-on-Chip* (NoC), que permite a comunicação entre os diferentes componentes. Essa comunicação é feita utilizando como referência o modelo *Open System Initiative* (OSI). As arquiteturas GALS e NoC serão explicadas com mais detalhe no capítulo seguinte.

O projeto IOPT-TOOLS, que se encontra em fase de desenvolvimento, permite a utilização de modelos expressos em redes de Petri *Input Output Place Transition* (RdP-IOPT) para especificação do comportamento do sistema. Neste projeto estão a ser desenvolvidas várias ferramentas para edição de RdP-IOPT, tal como geração do código C, geração do espaço de estados e geração de código para sistemas GALS e NoC, que vai ser realizado neste trabalho e incluído neste projeto. O projeto IOPT-TOOLS irá ser explicado com mais detalhe no capítulo seguinte.

1.2 Objetivos

O objetivo principal é o de desenvolver um protótipo de ferramenta que permita a geração automática de um ficheiro de configuração para plataformas específicas através de redes de Petri (RdP) permitindo a integração de um conjunto de componentes previamente definidos.

Uma vez que os modelos RdP se encontram descritos no formato PNML (*Petri Net Markup Language*), a ferramenta a desenvolver deverá identificar os eventos de entrada e saída de cada componente, bem como permitir escolher a plataforma em que cada componente deve ficar, permitindo que o utilizador defina a forma de interligação entre os eventos: se utilizando *Bus*, *Ring*, *Double Ring*, *Matrix*, Toroidal ou *Wrappers* ou através de ligação direta (*Serial*).

1.3 Estrutura

Este documento, para além deste capítulo introdutório, onde foi feito um enquadramento do assunto discutido nesta dissertação e onde foram apresentados os objetivos, é constituído por mais 6 capítulos.

No capítulo dois, apresentam-se os conceitos e as tecnologias que fundamentam o trabalho. Visto que o trabalho desenvolvido tem em conta resultados apresentados em outras dissertações, faz todo o sentido fazer uma breve apresentação dos sistemas GALS como das *Network-On-Chip*.

O capítulo três, incide sobre as soluções para as quais a ferramenta desenvolvida se destina, descrevendo as várias soluções de suporte à comunicação que foram utilizadas na realização da ferramenta desenvolvida.

O capítulo quatro, incide sobre a descrição do sistema proposto, ou seja, apresentação dos requisitos e especificações da ferramenta que vai permitir a geração do ficheiro de configuração do sistema a implementar, bem com as considerações tomadas.

O capítulo cinco, apresenta um exemplo de aplicação da ferramenta, onde é apresentado todo o fluxo de desenvolvimento, desde a modulação do sistema, até à obtenção do

ficheiro necessário para a implementação do sistema.

Por fim o capítulo seis, são apresentas as conclusões sobre o trabalho realizado, mostrando as dificuldades encontradas e também uma avaliação ao trabalho realizado. Apresenta-se ainda possibilidades de trabalhos futuros com o objetivo de melhoramento da ferramenta apresentada.

2

Fundamentos e tecnologias

Neste capítulo, serão abordados alguns conceitos teóricos necessários para o melhor entendimento do trabalho, bem como tecnologias utilizadas. Será apresentada a norma XML, bem como as origens, o comportamento, as vantagens e a estruturadas das RdP, e também as RdP-IOPT e a norma PNML. Será feita uma breve descrição do funcionamento das GALS e das *Network-on-Chip*.

2.1 XML

Extensible Markup Language (XML) é um formato de texto simples e flexível que deriva da SGML (*Standard Generalized Markup Language*). A XML é uma especificação técnica desenvolvida pela W3C (*World Wide Web Consortium*) para superar as limitações do HTML, definido como padrão das páginas Web. Fornece também um formato normalizado para documentos de computadores que é flexível o suficiente para ser personalizado para domínios tão diversos como *websites*, intercâmbio de dados eletrónicos, etc. [15]. As mais valias do XML, que permitem que dados sejam partilhados entre utilizadores, são:

- Sintaxe simples, fácil de gerar e analisar;
- Facilidade de analisar e retificar;
- Independência da linguagem e da plataforma.

O formato de um ficheiro XML é apresentado na Figura 2.1, onde é possível verificar que é composto por um *element* chamado *person*. Este elemento é delimitado pela a *star-tag* <person> e pela *end-tag* </person>. Tudo entre *star-tag* e a *end-tag* de *element* é

chamado de conteúdo do *element*. O conteúdo do *element* apresentado na Figura 2.1 é "Alan Turing" [14]. Na Figura 2.2 podemos ver um ficheiro XML mais complexo.

```
<person>
  Alan Turing
</person>
```

Figura 2.1: Descrição de um XML num formato muito simples [14]

```
<person>
  <name>
    <first_name>Alan</first_name>
    <last_name>Turing</last_name>
  </name>
  <profession>computer scientist</profession>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
</person>
```

Figura 2.2: Documento XML mais complexo, descrevendo uma pessoa [14]

2.2 Redes de Petri

As redes de Petri são um formalismo gráfico que permite a descrição e análise de processos concorrentes que se encontram em sistemas com muitos componentes. O conceito de Redes de Petri tem a sua origem na tese de doutoramento *"Communication with Automat"* de Carl Adam Petri, submetida em 1962. Desde essa altura, que as RdP têm vindo a ser desenvolvidas e usadas, tanto em áreas teóricas como práticas. O trabalho realizado por C. A. Petri, chamou a atenção A. H. Holt, que conjuntamente com outros investigadores, desenvolveu a muita da teoria inicial, notações e representação gráfica das redes de Petri, [24].

2.2.1 Estrutura e comportamento da Rede de Petri

Com o trabalho publicado em [24], uma rede de Petri é um tipo particular de um grafo direcionado e bipartido, constituído por dois tipos de nós, chamados lugares e transições, e por arcos que ligam lugares a transições ou transições a lugares.

Na representação gráfica, os lugares são representados por círculos e as transições por barras ou retângulos. Os arcos podem ser classificados com um peso associado, representado por um número inteiro positivo, caso o arco tenha o peso de "1" esse valor pode ser omitido. Cada lugar pode conter uma ou mais marcas, que são representadas por um pequeno círculo denominado marca.

Na modelação são utilizados os conceitos, condição e evento, em que os lugares representam as condições e as transições, os eventos. Uma transição (um evento) pode ter um certo número de lugares de entrada, bem como um certo número de lugares de saída, representando as pré-condições e as pós-condições do evento, respetivamente. A presença de um marca num lugar significa que a condição associada a esse lugar é verificada.

Uma definição formal de uma RdP é dada por uma expressão com 5 parâmetros, PN = (P, T, F, W, M_0) onde, [24]:

P = $\{p_1, p_2, \ldots, p_3\}$ é um conjunto finito de lugares,
T = $\{t_1, t_2, \ldots, t_3\}$ é um conjunto finito de transições,
F (P $\times$ T) (T $\times$ P) é um conjunto finito de arcos,
W: F $\rightarrow$ {1, 2, 3, ...} função dos pesos atribuídos aos arcos,
M_0: P $\rightarrow$ {0, 1, 2, 3, ...} função da marcação inicial.

De forma a ilustrar as definições acima apresentadas, considera-se as RdP da Figura 2.3, que tem três lugares denominados p_1, p_2 e p_3 e tem apenas uma transição denominada t_1.

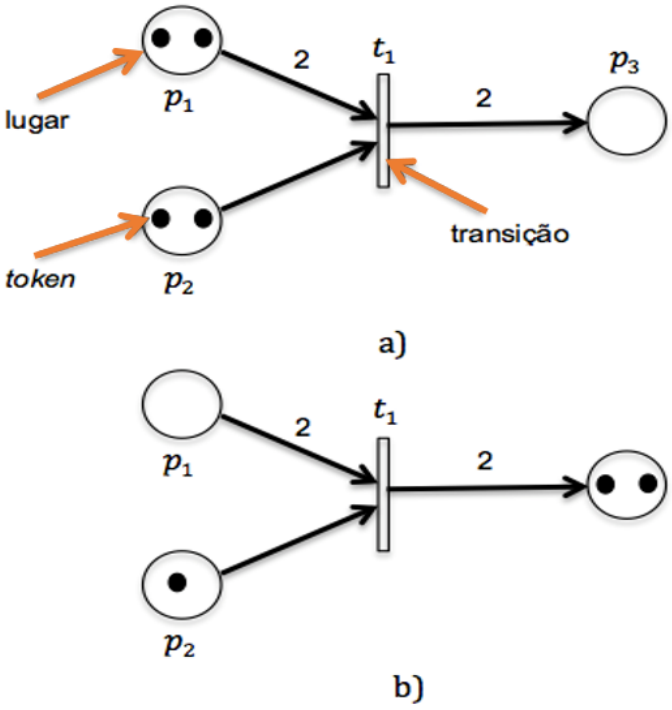


Figura 2.3: Execução de um RdP: a) antes do disparo b) depois do disparo

Para a simulação de um sistema dinâmico utilizando redes de Petri é necessário que uma marca ou um estado mudem de acordo com as seguintes regras de transição:

- a transição encontra-se habilitada se cada lugar de entrada estiver marcado com um número de marcas maior ou igual ao peso do arco que o liga à transição;
- Uma transição só dispara se o evento tiver ocorrido;

- Caso aconteça um disparo, o número de marcas removidas dos lugares de entrada será igual ao peso dos arcos associados, sendo adicionadas marcas nos lugares de saída de acordo com o peso dos arcos de saída.

As regras de transição mencionadas anteriormente, podem ser verificadas através do exemplo demonstrado na Figura 2.3.

2.2.2 Redes *Input-Output Place-Transition*

Nesta secção é apresentada a classe de redes de *Petri Input-Output Place-Transition* [13]. Esta classe é definida como uma extensão à classe de redes de Petri Lugar-Transição [24], permitindo a modelação de controladores, bem como a execução autónoma do modelos por um controlador. Quando comparadas com as redes P/T, as redes IOPT têm características que permitem modelar o comportamento dos controladores e também para suporte de geração automática de código. São elas:

1. Sinais de entrada e saída;
2. Eventos de entrada e saída;
3. Arcos de teste com um peso associado;
4. Prioridades associadas a transições;
5. Cada transição pode ser associada eventos de entrada e eventos de saída;
6. Cada lugar deve ter um atributo permitindo a identificação do máximo número de marcas que podem estar presente nesse lugar;
7. Identificação de conjuntos de transições em conflito;
8. Identificação de conjuntos de transições síncronas.

2.2.3 Petri Net Markup Language

Com base no que é proposto em [4], a *Petri Net Markup Language* (PNML) foi projetada como um formato de representação de RdP, baseado em XML (*eXtensible Markup Language*), permitindo o intercâmbio de formatos dessas RdP, independentemente das ferramentas e das plataformas utilizadas. O *design* da PNML, é definido por alguns princípios, são eles:

- Flexibilidade;
- Compatibilidade;
- Não ambiguidade.

A flexibilidade da PNML significa que esta deve poder representar qualquer tipo de RdP, com extensões e características específicas.

A compatibilidade significa que tanta informação quanto possível pode ser trocada entre diferentes tipos de RdP. De maneira a atingir essa compatibilidade, a PNML tem vindo a desenvolver acordos para definir uma etiqueta com um determinado significado. É no *Conventions Document* que toda a sintaxe e o significado de todas as extensões são predefinidos.

A ambiguidade é removida do formato, assegurando que a RdP original e o seu tipo particular, possam ser unicamente determinados pela sua representação PNML. Com este intuito, o PNML suporta a definição de diferentes tipos de RdP.

2.2.4 Operação Net splitting

A operação de *Net splitting*, apresentada em [6], é baseada em definir um conjunto de corte válido, isto significa definir um conjunto de nós através do qual é possível dividir a rede em vária sub-redes. A cada nó pertencente ao grupo de corte, apenas se pode aplicar uma de três regras, que foram definidas dependendo do tipo de nó e também dos arcos de entrada. A primeira regra é, no caso de o nó de corte ser um lugar, nas duas outras o nó de corte é uma transição. Quando o nó de corte é uma transição, são aplicadas duas regras, numa delas os arcos de entrada da transição são apenas de um componente, na outra regra os arcos de entrada da transição vêm de componentes diferentes.

A operação de *Net splitting* proporciona a divisão de modelos de RdP, o que resulta na geração de vários submodelos. Para resolver o problema de comunicação entre esse sub-modelos é proposto o uso canais de comunicação síncronos, anexados às transições.

A Figura 2.4 mostra o exemplo onde a primeira regra (lugar como nó de corte) é aplicada.

- figura 2.4a) representa uma RdP onde *P1* é o elemento do conjunto de corte (*CS*);
- figura 2.4b) mostra o resultado da remoção do lugar *P1*, onde são obtidos três componentes em três regiões diferentes;
- figura 2.4c) representa o resultado da operação *splitting*, em que no componente um se encontra a transição um e quatro como *slaves* e no componente dois e três são apresentadas com *masters*.

Neste exemplo são usados dois conjuntos síncronos, um deles é composto pela transição *T1* (*master*) no componente dois, e *T1_copy* (*slave*) no componente um. A transição *"slave"* não tem arcos de entrada, a única condição de disparo é imposta pelo disparo da transição *"master"*.

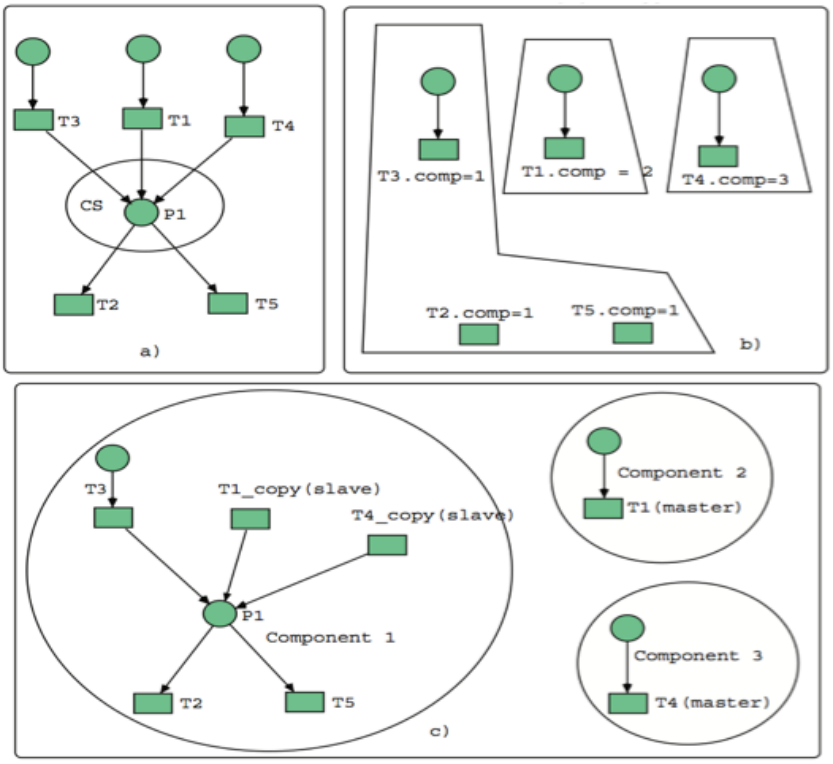


Figura 2.4: Divisão da RdP através do lugar [6]

Considerando agora que o nó de corte é uma transição, existem duas regras para a sua aplicação, que são apresentas na Figura 2.5 e na Figura 2.6.

A RdP apresentada na Figura 2.5a) tem como nó de corte a transição *T1*. A regra "os arcos de entrada da transição são apenas de um componente" é mostrada no exemplo na Figura 2.5b), onde são obtidas dois componentes identificas em duas regiões diferentes. No componente um, verifica-se que os arcos de entrada da transição *T1* fazem parte deste componente e também é possível verificar que foi removido o nó de corte, a transição *T1*. Na Figura 2.5c) é mostrado o resultado da operação *splitting*.

Na Figura 2.6 a regra aplicada mostra que os arcos de entrada da transição *T1*, onde *T1* é o nó de corte, vêm de componentes diferentes. Isso é possível verificar-se na Figura 2.6b). A Figura 2.6a) representa uma RdP onde *T1* é o nó de corte. A Figura 2.6c) mostra o resultado da operação *splitting*.

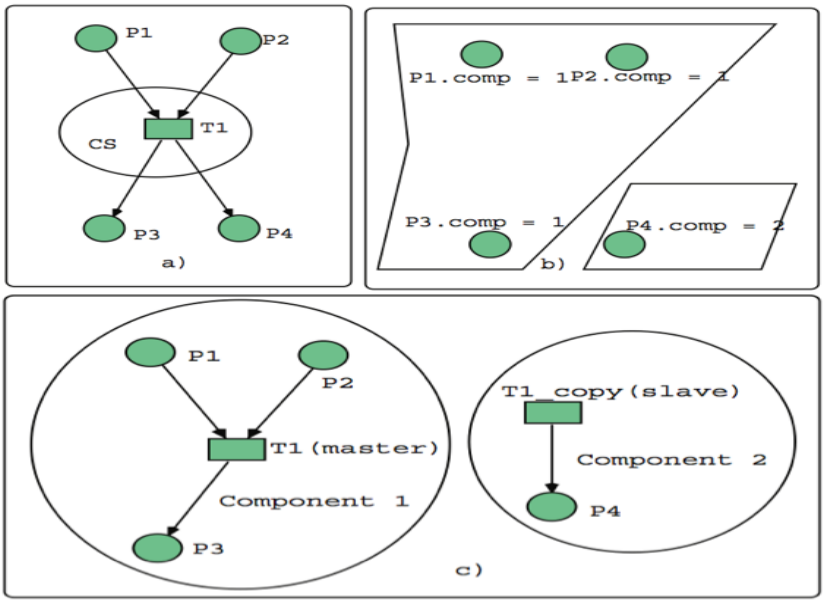


Figura 2.5: Divisão da RdP através da transição [6]

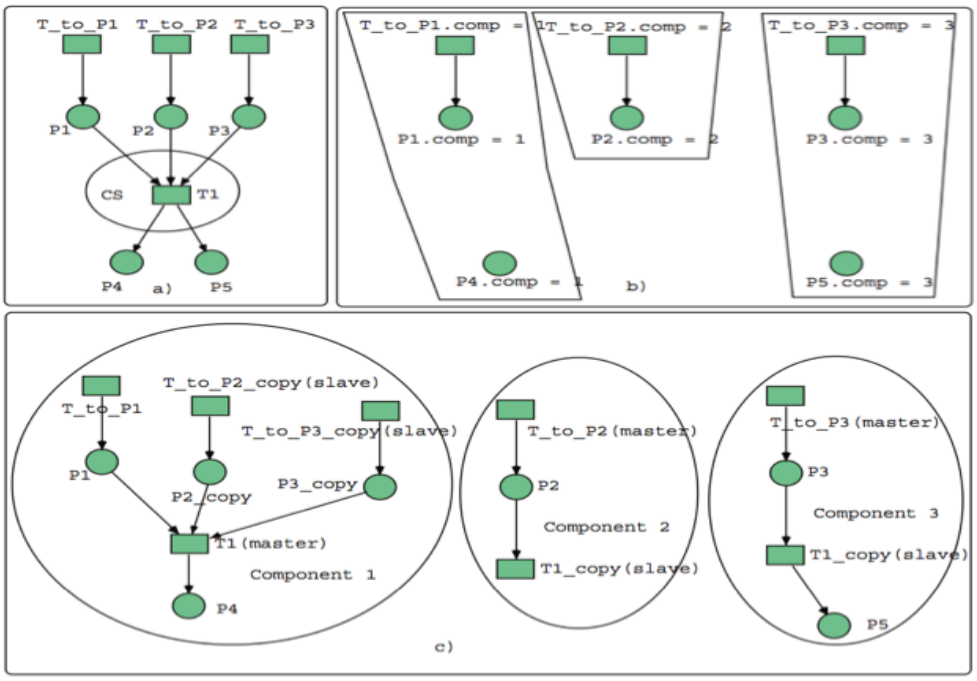


Figura 2.6: Divisão da RdP através da transição [6]

2.3 Sistemas Síncronos

Baseado em [9], um circuito síncrono é um circuito digital onde todas as suas partes são sincronizadas com um sinal de *clock*. Um sistema sem *clock* global é considerado assíncrono, o sistema assíncrono é discutido de seguir.

O estado interno muda apenas quando ocorre uma alteração, positiva ou negativa do *clock*. Isto leva a que todas as operações no sistema necessitem de ser realizadas e completas entre dois ciclos de *clock*. Se este critério for satisfeito, o sistema é considerado fiável e todo o comportamento do circuito pode ser previsto com precisão. Cada operação lógica introdução um atraso no sistema, o que na pratica, limita a velocidade máxima a que o sistema síncrono pode ser executado.

Este tipo de sistema tem duas desvantagens muito significativas, são elas:

- O sinal de *clock* é distribuído para todos os circuitos *flip-flops* simultaneamente. O sinal de relógio é um sinal de alta frequência que, potencialmente, consome grandes quantidades de energia. Mesmo que inativo, os *flip-flops* consomem alguma energia, contribuindo desnecessariamente para um desperdício de energia e acumulação de calor.
- A velocidade máxima de *clock* é limitada ao caminho mais longo do circuito. Isto significa que tanto uma operação complexa como uma operação simples, têm de ser executadas num ciclo de *clock*.

Considerando um sistema globalmente sincrónico, constituído por vários módulos, em que cada módulo é projetado para funcionar numa determinada velocidade de *clock*, encontrar um *clock* global que permita a comunicação entre os diferentes módulos, revela ser um tarefa muito árdua.

Outro problema desta solução é a não garantia do mesmo tempo de chegada do sinal de *clock* a todos os componentes do circuito, devido aos atrasos nos fios. Se no passado o fator que limitava os circuitos eram os transístores, no presente o atraso nas propagações consomem uma larga parte do período de *clock*. Por outro lado, este tipo de circuitos é bastante utilizados na indústria, por serem amplamente estudados e terem um grande apoio da ferramentas CAD (*Computer Aided Design*).

2.4 Sistemas Assíncronos

Sistemas assíncronos não são regidos, quer localmente quer globalmente, por um sinal de *clock*. Os circuitos usam *handshaking* entre os seus componentes, a fim de conseguirem a sincronização necessária, comunicação e sequência de operações [27]. Estas diferenças têm vantagens relativamente aos sistemas síncronos pelas seguintes razões:

- Baixo consumo de energia, devido ao consumo ser zero quando o sistema se encontra em pausa;
- Alta velocidade de operação;
- Menor emissão de som eletromagnético;
- Robustez para variações no fornecimento de energia e temperatura.

As principais desvantagens que este tipo de circuitos enfrenta, não são tanto intrínsecos à tecnologia em si, mas devido a uma inércia da comunidade em mudar de uma interface síncrona para uma nova interface. Somando-se a isso, também, a falta de desenvolvimento e ferramentas de teste.

2.5 Interface Síncrona-Assíncrona

O principal problema de interação entre os diferentes domínios de sincronização, surge fundamentalmente da diversidade dos sinais de transição. No domínio assíncrono, não existe um intervalo de tempo associado a sinais de transição, logo, o comportamento meta estável em *flip-flops* periféricos de módulos localmente síncronos é mais provável de ocorrer.

O esquema convencional para resolver este tipo de problemas é o uso de sincronizadores. Isto inclui o mecanismo de *double-latching* e também *pepiline synchronization*. Embora estes métodos reduzam a probabilidade de mau funcionamento, este tipo de métodos não o exclui. Também adiciona latência indesejada a cada comunicação. Isto tudo faz com o sistema seja sujeito a falhas, e também indesejado para lidar com o paradigma que atravessa os limites entre o síncrono e o assíncrono [25].

2.6 GALS

Os sistemas GALS (*Globally Asynchronous Locally Synchronous*) foram proposto inicialmente em 1984, por D. M. Chapiro [5] na sua tese de doutoramento, mas devido à impraticabilidade do *design* na altura, apenas em meados de 90 e no início de 2000, começaram a surgir propostas práticas, tais como introdução do *pausible* ou *stretching clocking*. Estas soluções tinham como objetivo, reduzir o consumo de energia e o *overhead* e melhora o desempenho do sistema.

Os sistemas GALS combinam os benefícios dos sistemas síncronos e assíncronos. Os sistemas GALS contêm vários blocos síncronos independentes, que operam com o seu *clock* local e comunicam assincronamente entre si. A principal característica deste sistema é a ausência de um *clock* global, utilizando diferentes *cloks* locais, eventualmente em diferentes frequências de funcionamento.

Para representação das GALS é necessário uma extensão das RdP IOPT através dos *Asynchronous-Channel* (ACs) e *Time Domains* (TDs). A introdução destes dois componentes permite tornar um sistema globalmente síncrono num sistema GALS. Os *asynchronous channels* representam a interação assíncrona entre componentes com diferentes domínios temporais, enquanto os *time domains* são usados para associar cada nó (lugar e transição) a um diferente componente, em que cada componente é localmente síncrono, [22]. Um exemplo de extensão de um sistema IOPT com *Assynchronous-Channel* e *time domains* é apresentado a seguir.

Figura 2.7: Extensão da RdP IOPT com *Asynchronous-Channel* e *Time Domains* especificando um sistema GALS com dois componentes [22]

2.7 Wrapper

Para que uma LSI (*Locally Synchronous Island*) comunique com outra LSI, foi criada uma interface assíncrona, que pode ser observada na Figura 2.8. A interface assíncrona mostra a estrutura do *wrapper* assíncrono que foi considerado para resolver o problema de sincronismo entre LSIs [9]. O *wrapper* assíncrono é constituído por: um *Input Port*, uma *FIFO buffer* e um *Output Port*. A utilização dos *Ports* serve para fazer a transição de um sistema LSI para o *FIFO buffer* e novamente para outro LSI. Cada uma das partes do *wrapper* será explicada com maior detalhe nas seguintes secções.

Na Figura 2.8 podemos ver que o *wrapper* assíncrono tem como entradas: dois *clock's* são eles o *TX_clock* e o *RX_clock* e um sinal de dados representado por *Sin*. Como saída temos o *Sout* que é um sinal de dados. O *wrapper* servirá para interligar os diferentes eventos de cada componente [9]. Os componentes podem ser obtidas utilizando a ferramenta SPLIT, que foi concebida para o projeto FORDESIGN, ou através do projeto IOPT-Tools, explicado no Capitulo 2.7.

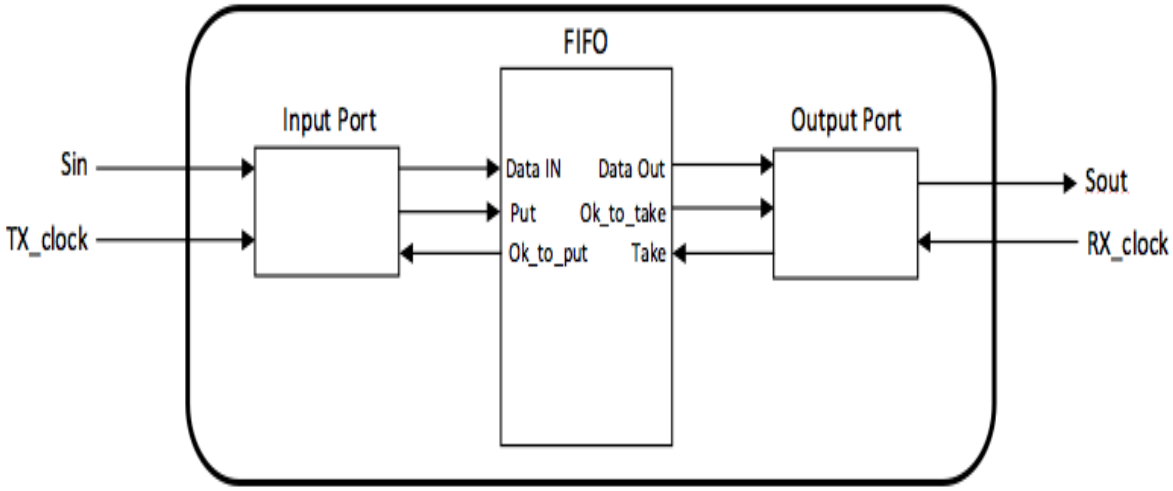


Figura 2.8: Interface do Wrapper Assíncrono [9]

2.7.1 Input Port

Neste capitulo será feita uma breve descrição do funcionamento do *Input Port*, apresentado em [9]. A Figura 2.9 mostra a sua composição e o seu funcionamento é o seguinte:

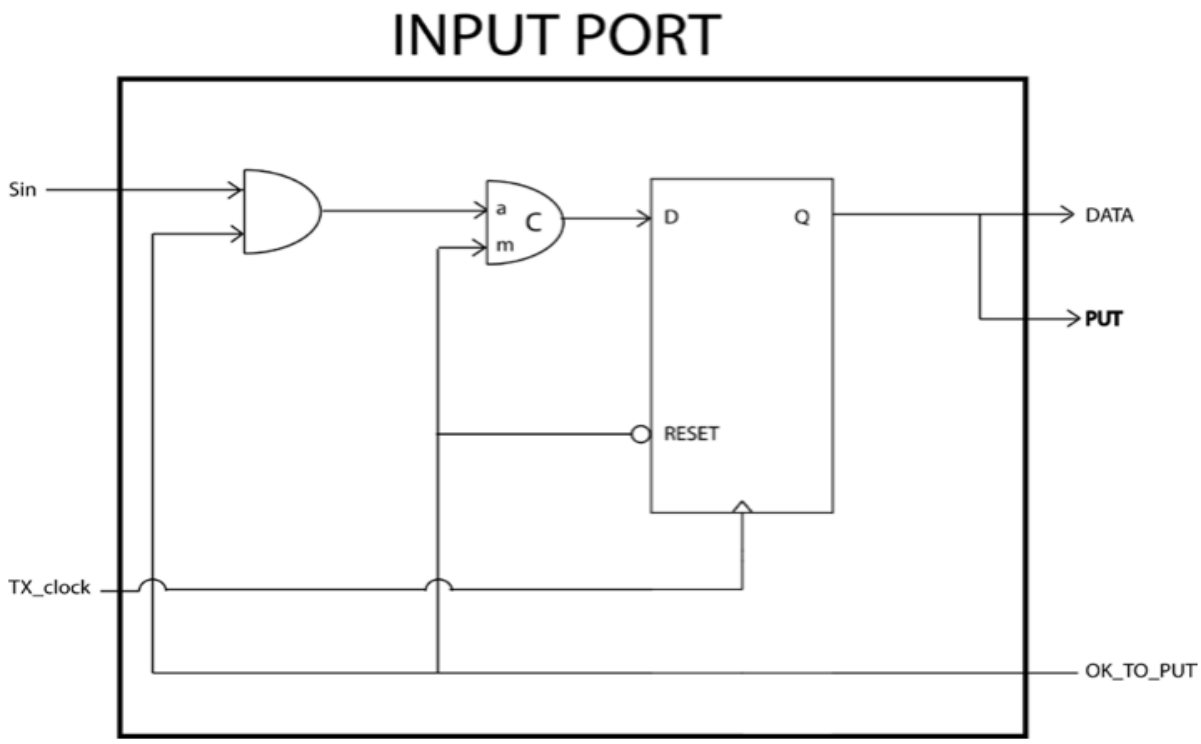


Figura 2.9: Input Port [9]

Se a *FIFO* está em condições de aceitar novas entradas então *ok_to_put* está ativa. Se um evento ocorrer, *Sin* estará ativo durante um certo período de tempo, logo o primeiro *AND* estará elegível para que um valor possa passar. Caso o *AND* esteja ativa, *C Minus Element* estará também ativo, enquanto *Sin* e *ok_to_put* estiverem ativos. O *latch D* está representado para estender o sinal até haver uma confirmação que este entrou na *FIFO buffer*.

Na Figura 2.10 é retratado o *C-Element*. O seu funcionamento é o seguinte:

Quando as duas entradas são 0, a sua saída será 0, quando ambas as entradas forem 1, a sua saída será 1, para quaisquer outras combinações na entrada, a saída não se altera. Assim, pode-se concluir que, se a saída mudar de 0 para 1, as duas entradas são 1, e caso a saída mude de 1 para 0, pode concluir-se que as duas entradas são 0.

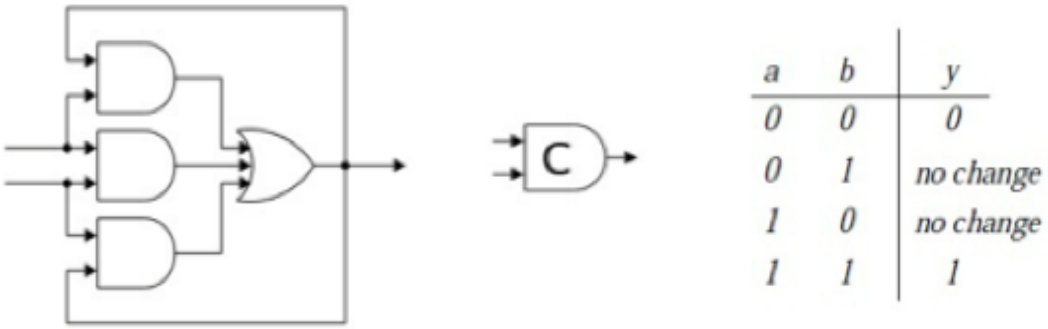


Figura 2.10: C-Element: Possível Implementação, Símbolo e Tabela de Verdade [9]

Uma variação do *C-Element*, denominada *C-Element assimétrico*, permite que as entradas, apenas afetem a operação do elemento, quando transitam numa direção. As entradas assimétricas são anexadas ao menos (–) ou ao mais (+). As entradas comuns, que afetam ambas as transições estão ligadas ao centro do símbolo. Quando transitam de zero para um o *C-Element* terá em conta a entrada comum e a entrada assimétrica (+). Todos as entradas devem estar ativas para que a transição fique ativa. Ao considerarmos a transição de um para zero o *C-Element* vai ter em consideração a entrada comum e a entrada assimétrica (-). Todas as entradas devem estar inativas, para que a transição fique inativa. A Figura 2.11 representa o *C-Element assimétrico*. Os sinais (+) de entrada são representados por um 'P', os sinais (-) de entrada são representados por um 'M' e as entradas comuns são representados com um 'C'.

Os sinais *Put* e *Data* serão a saída do *flip-flop D*, sempre que a saída de *C Minus Element* e *TX_clock* estejam ambos ativos. Desta forma o *Put* está sincronizado como o *clock* do transmissor e naturalmente com a *Data* a ser transmitida. Portanto a *Data* transmitida será inserida na *FIFO* ao mesmo tempo que o *clock* for ativado, com um certo atraso introduzido no *Flip-Flop D*.

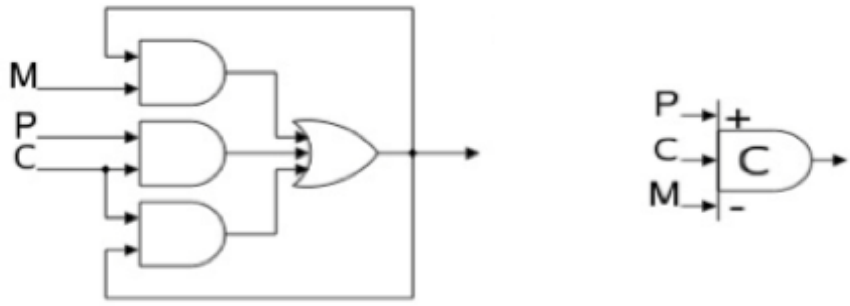


Figura 2.11: C-Element assimétrico: Possível Implementação, Símbolo [9]

Quando a *FIFO* não estiver em condições de receber mais dados, o *ok_to_put* ficará inativo. Isto implica, um *reset* ao *C Minus Element*, assim como o *latch D*. Assim que a *FIFO* estiver pronta para receber novos dados, todos os elementos estarão livres para receber novos valores.

2.7.2 Output Port

O *Output Port*, apresentado em [9], será descrito neste Capítulo. A Figura 2.12 mostra a composição do *Output Port* utilizado neste trabalho. O sinal *Data* precisa de ser sincronizado com o *clock* de recetor para evitar a meta estabilidade. Isto é conseguido através do uso de dois *Flip-Flops D*, como é verificado na Figura 2.12. Uma breve explicação do funcionamento do *Output Port* é feita a seguir.

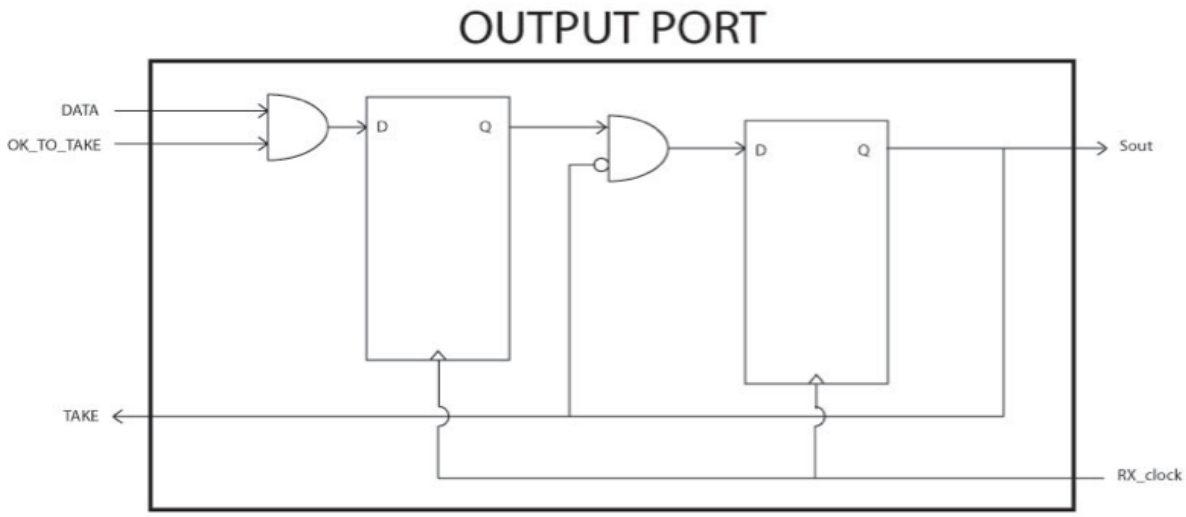


Figura 2.12: Output Port [9]

A porta AND encontra-se antes dos dois *Flip-Flop D*, para que a *Data* e o *ok_to_take* estejam

ambos presentes, antes de entrarem no sincronizador.

Take será ativo assim que *Sout* esteja ativo. Isto implica que a sincronização foi um sucesso e o fluxo dentro do sincronizador pode ser interrompido, senão, o sinal será estendido por mais um *clock*. Para realizar isto foi introduzido uma porta lógica *AND* entre os dois *Flip-Flops*. O *AND* só é ativo se *Sout* for desativado.

2.7.3 FIFO Buffer

A *FIFO (First In First Out) buffer*, apresentada em [9] não é mais que uma sequência de *latchs* que guarda dados em ordem, o último *latch* guarda o primeiro item de dados, enquanto o primeiro guarda o ultimo item de dados. Falando em termos práticos, o primeiro item a entrar é o primeiro a sair. A *FIFO buffer* utilizada neste trabalho é representada na Figura 2.13.

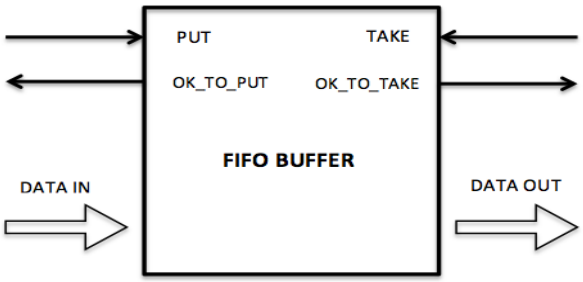


Figura 2.13: FIFO Buffer com os sinais de entrada e saída

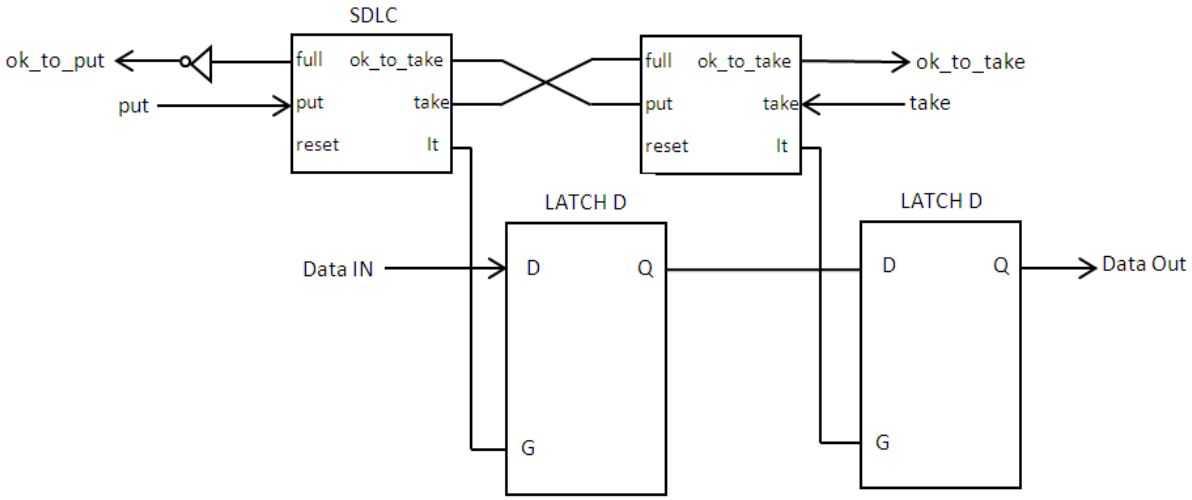


Figura 2.14: *FIFO* do *wrapper assíncrono*

A *FIFO Buffer* da Figura 2.13 possui dois *semi-decoupled latch controllers* (SDLC), um para

cada *latch*. Cada *latch* aceita apenas um bit. Na Figura 2.14 são apresentados dois SDLC, mas esse número pode variar consoante a necessidade de aumentar ou diminuir a dimensão da *FIFO Buffer*, o que será explicado com maior detalhe no capítulo seguinte.

A arquitetura do SDLC que está representada na Figura 2.15 a), é constituída por um *cnminus* e por um *cplus*, que se encontram representados na Figura 2.15 b) e c) respetivamente. A saída do *cplus* só estará ativa quando o sinal *p* e *a* estão ativos e continua ativo, enquanto o sinal *a* está activo. O *cnminus* tem um comportamento idêntico ao C-Element utilizado no Input Port, exceto a saída que fica inativa quando o sinal *m* fica inativo.

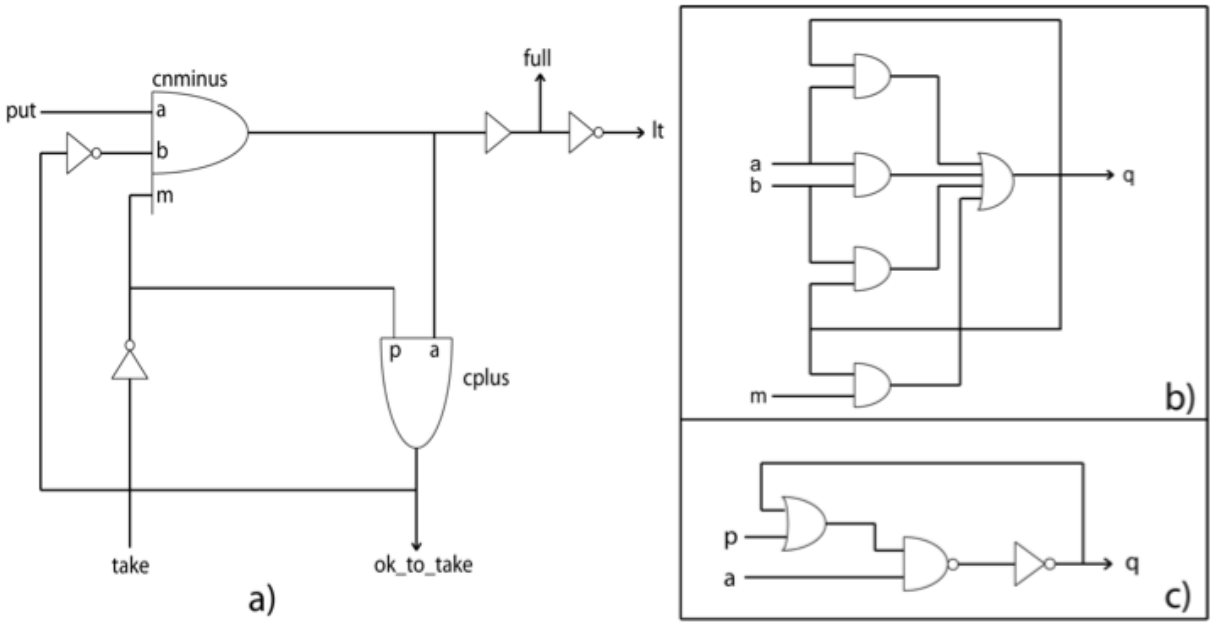


Figura 2.15: A) Constituição do *semi-decoupled latch controller*, B) *cnminus*, C) *cplus*

2.8 Network-on-Chip

Com base no que é apresentado em [26], as *Network-on-Chip* (NoCs) representam uma nova categoria para as comunicações *on-chip*, onde a abstração dos protocolos em camada é utilizada para modular a comunicação. Isto implica que a computação e a comunicação estejam separadas entre si.

A ideia fundamental das NoCs é a aplicação de camadas, tal como nas telecomunicações e nas redes de computadores. O modelo *Open System Initiative* (OSI) utilizado não é necessariamente seguido em rigor, mas sim adaptado consoante as necessidades. Diferentes funcionalidades de uma NoC podem ser mapeados para uma ou mais camadas do modelo OSI. A rede pode utilizar dois tipos de ligação, são elas: orientado à ligação

(comutação de circuitos) e não orientado à ligação (comutação de pacotes). Uma interface apropriada, é importante para a formação dos pacotes ou para estabelecer a ligação de comunicação. A NoC é composta por nós que encaminham o tráfego, esses nós chamados comutadores, que ligam qualquer entrada a qualquer saída, podem também conter *buffers* para guardar os pacotes, que passam por esses nós. A topologia utilizada determina como os recursos da rede se interligam. Muitas das abordagens utilizam as seguintes topologias, malha (*mesh*), malha toroidal, anel bidirecional, octogonais e em árvore [10].

As propostas de topologias que melhor representam os NoCs, apresentas em [26], são as seguintes:

- *xPipes* – uma rede flexível constituída pela parametrização de componentes sintetizáveis. A topologia pode ser especificada pelo designer. São utilizadas tabelas de encaminhamento estáticas para a comutação dos pacotes. A rede é síncrona e, quer os comutadores quer as ligações, são *pipelined*, para atingir uma elevada taxa de transmissão. Um pacote de reconhecimento é reenviado ao transmissor, após sucesso da respetiva transmissão.

- *Proteo* – é muito similar ao *xPipes* em muito aspetos. As implementações são baseadas em interligações parametrizadas de blocos *Intellectual Propriety* (IP), e a topologia pode ser escolhida pelo designer. Contudo, as implementações iniciais tinham uma tipologia em anel, hierarquicamente particionadas em anéis mais pequenos se necessário. Existe a possibilidade de se utilizar pacotes de confirmação, mas a implementação da retransmissão e do erro de correção, estão a cargo do designer.

- *Nostrum* – utiliza a topologia em malha, como recursos colocados nos núcleos ligados por uma matriz de interligação. O modelo OSI é amplamente explorado, mas apenas é obrigatório a utilização das três camadas mais baixas do modelo (física, lógica e de rede). Algum controlo de erros é feito na camada lógica. Para melhor o circulação de tráfico na rede, um circuito virtual pode ser formado, para que uma fração da largura de banda total possa ser alocada a serviços prioritários.

- *SoCbus* – utiliza uma topologia em malha 2D. O objetivo é mudar o barramento embutido por uma rede de comutação de circuitos, para fornecer uma largura de banda mais elevada. Existe uma única *Central Coordination Node* (CNN) que realiza as funções de coordenação do sistema. O CNN gera a configuração de cada nó quando um circuito é iniciado. Não existe controlo de fluxo, mas um sinal de confirmação é fornecido.

- *SPIN fat tree* – Ao contrário de outros NoCs, o *Scalable Programmable Integrated Network* (SPIN), tem como topologia de referência uma *fat-tree*. Uma *fat-tree* é uma estrutura em árvore com encaminhadores nos nós e recursos computacionais nas suas folhas, exceto nos nós que têm "pais" repetidos. Não é considerada a deteção de erros bem como a retransmissão.

- *XGFT* – As *Generalized Extended Fat Trees* (XGFTs) podem ser construídas, através de nós que permitem o encaminhamento de pacotes, no sentido ascendente e descendente, através de blocos de comutação separados. Foi provado que a performance da solução XGFT é superior à solução em malha.
- *Redes CDMA* – A *Code-Division Multiple Acess* (CDMA) é uma técnica utilizada para comunicações sem fios *spread-spectrum*. É também aplicada em barramento embutido. Um árbitro centralizado é necessário para configurar a interface de destino para receber o código do canal desejado.
- *Philips Æthereal* – Uma NoC Æthereal utiliza encaminhamento sem contenção, ou *pipelined time-division-multiplexed circuit switching*, para garantir serviços de desempenho garantido. Embora todos os dados tenham a mesma prioridade, eles podem obter diferentes reservas de largura de banda. Os *slots* de tempo e o encaminhamento são "programados" usando tabelas que se encontram em cada nó. Existem dois modelos de programação, distribuído e centralizado.

O vasto número de parâmetros dos NoCs é um problema inerente do design e da comparação, por exemplo na definição da topologia, comutação, controlo de fluxo e algoritmo de encaminhamento [8].

No caso geral, não existem NoCs ideais. Contudo, de acordo com avaliação que é feita para a aplicação dos NoCs, é possível identificar os parâmetros mais significativos, para certos cenários de aplicação. De seguida serão apresentadas as principais propriedades dos NoCs, são elas [8]:

- Separar as comunicações da computação;
- Evitar comunicações globais e controladores centralizados;
- Permite um número arbitrário de terminais;
- Permite múltiplas voltagens e diferentes frequências;
- Oferece várias garantias de transferência de dados;
- Oferece suporte para testes de sistema;

Certas propriedades, tais como o tempo de execução, consumo de energia, área de silício e a latência, são as métricas mais frequentes e importantes e, naturalmente, estas propriedades devem ser reduzidas.

Tendo em conta que o objetivo é o de caracterizar as NoCs e a sua comunicação entre NoCs quer seja na mesma plataforma quer em plataformas distintas. Assim sendo são analisadas duas métricas. Facilidade de interligação entre plataformas heterogéneas e

flexibilidade de parametrização. Na facilidade de interligação entre plataformas heterogéneas será avaliado o hardware / software necessário, bem como a forma como os dados são transmitidos, pretende-se saber, se a arquitetura em questão se adequa e / ou é flexível aos requisitos que cada plataforma tem. A flexibilidade de parametrização tem como intuito, através dos vários componentes, pode vir a gerar-se a rede. Os parâmetros que definem uma NoC não devem ser rígidos para que não venha a ser necessário o projetista intervir de uma forma "manual".

2.8.1 Nó de Rede

O Nó de Rede, apresentado em [11] será descrito neste capítulo. Na Figura 2.16 é apresentada a sua estrutura, onde estão representados cinco módulos, que serão explicados posteriormente neste capítulo.

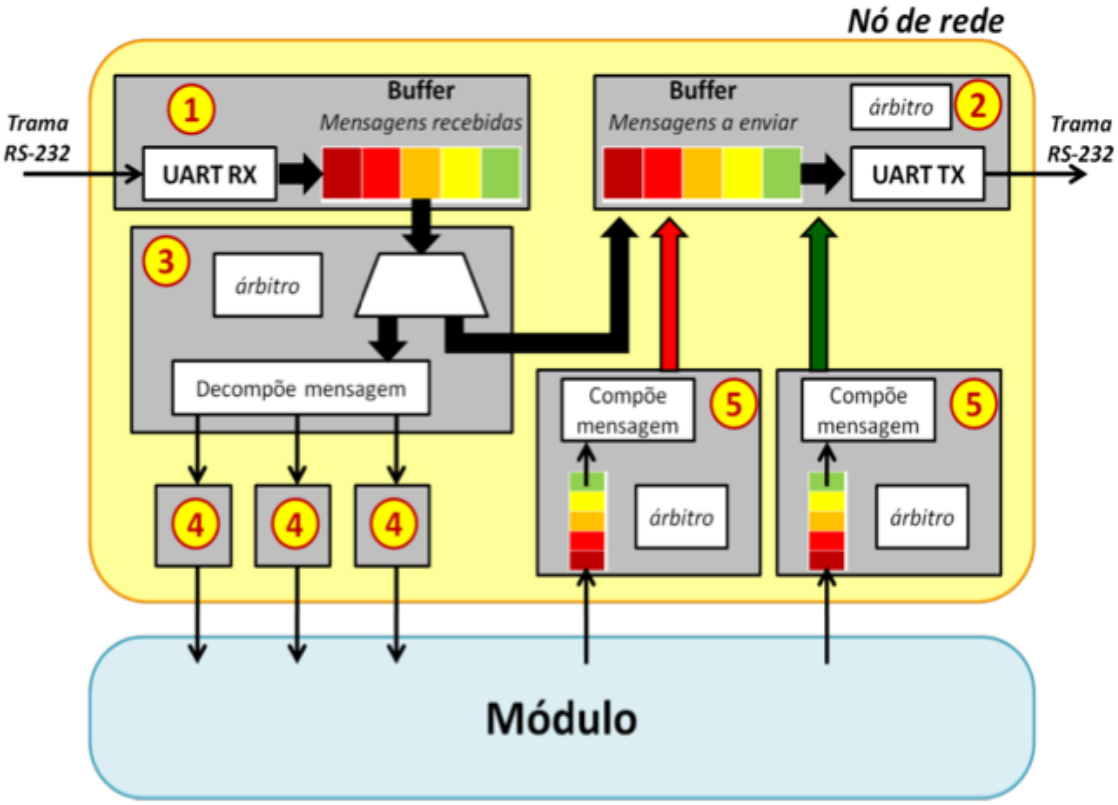


Figura 2.16: Nó de Rede [11]

Os cincos módulos representados são os seguintes:

1. Receção de tramas RS-232;
2. Envio de tramas RS-232;
3. Análise da mensagem recebida;

4. Ligação dos sinais e eventos de saída, com o código de execução associado ao modelo RdP-IOPT;

5. Ligação dos sinais e eventos de entrada, com o código de execução associado ao modelo RdP-IOPT.

O Nó de Rede tem o seguinte comportamento:

- Uma vez recebidas todas as tramas necessárias para a construção da mensagem, os *bytes* dessa mensagem serão retirados do *buffer*.
- O endereço de destino, que se encontra na mensagem é verificado, para ver se coincide com o endereço do nó. Se não coincidirem, significa que a mensagem não é destinada a esse nó mas sim a outro, assim a mensagem é colocada no *buffer* de saída (mensagens a enviar).
- Se o endereço de destino corresponde ao endereço do nó, significa que a mensagem se destina ao nó de rede, os sinais / eventos e respetivos valores são removidos da mensagem e colocados nos sinais específicos à entrada de cada modelo.
- Ao executar o modelo, as suas saídas serão colocadas num *buffer*. Quando este *buffer* não está vazio, é analisado e uma mensagem é composta e colocada no *buffer* de mensagens a enviar.

Através da Figura 2.16, é possível verificar que os diferentes módulos comunicam entre si. A comunicação entre os módulos, os *buffers* utilizados e todos os modelos será explicados nas secções seguintes.

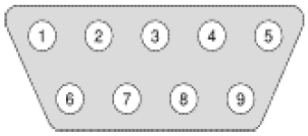
2.8.1.1 Protocolo RS-232

O RS-232 é um protocolo *single ended*, unidirecional (ponto-a-ponto), em que o sinal é referenciado à massa. Os *drivers* RS-232 adatpam a tensão do sinal RS-232 (-15 V a 15 V) para os níveis lógicos TTL (0 V a 5 V) .

A interface RS-232 está classificada para operar até 20 kbps. O desempenho do RS-232 vai ser influenciado pelo tamanho de cabo utilizado. Cabos muito longo requerem uma comunicação mais lenta, para poderem funcionar corretamente, enquanto cabos mais curtos permitem obter uma maior velocidade de comunicação.

Normalmente os dados são enviados em 7 ou 8 bits. Um bit de *start bit* marca o início da trama, após o *start bit* seguem-se os dados e, normalmente, também existe um bit para sinalizar o fim da trama (*stop bit*).

O RS-232 pode ser representado por um conetor de 25 pinos (DB 25) e também por um com 9 pinos, sendo este último o mais utilizado. Na Figura 2.17 é apresentado o conetor de 9 pinos [7].



Pino #	Nome do Sinal	Descrição do sinal
1	CD	Carrier detect
2	RXD	Receive data
3	TXD	Transmit data
4	DTR	Data terminal ready
5	GND	Ground
6	DSR	Data set ready
7	RTS	Request to send
8	CTS	Clear to send
9	RI	Ring indicator

Figura 2.17: Conector de 9 pinos [11]

2.8.1.2 Mensagem

Na Figura 2.18, é apresentado o formato da mensagem, que é constituída pelos seguintes elementos:

1. Origem – dados de origem da mensagem, necessário para se poder enviar a mensagem de reconhecimento;
2. Destino – dados de destino da mensagem, necessário para se saber qual o nó a que é destinada a mensagem;
3. Reconhecimento / Sinal – campo de um bit a designar se é ou não uma mensagem de reconhecimento;
4. # Sinal – código que cada nó tem associado aos sinais / eventos de entrada;
5. Valor – o valor que cada nó tem associado aos sinais /eventos de entrada;
6. Código – código de quatro bits para que se consiga identificar se a mensagem a circular na rede é fruto de uma falha e se esta já foi processada ou, se é uma mensagem nova do mesmo sinal com o mesmo conteúdo;
7. CRC – campo de verificação de redundância cíclica (CRC - *Cyclic redundancy check*);

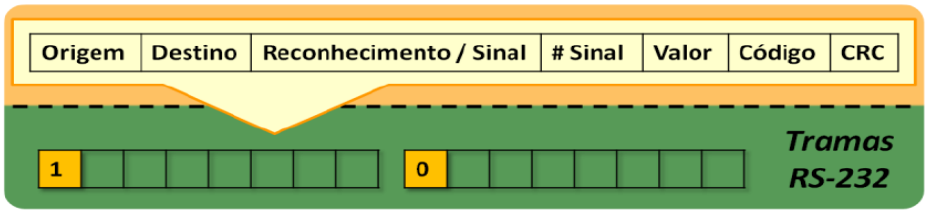


Figura 2.18: Formato da Mensagem [11]

A transmissão das mensagens é feita com base no protocolo lógico, utilizado na norma RS-232. As mensagens a transmitir irão ser compostas por múltiplos de 8 *bits*, mais um

de paridade, tendo assim um tamanho múltiplo de 11 *bits* (considerando *start* e *stop bit*). O tamanho será determinado pelo número de *bits* necessários para a codificação dos endereços dos nós de origem e de destino, da dimensão do sinal (ficando sempre o que tiver maior dimensão) e também do número de sinais existentes. O tamanho do CRC é definido conforme o número de *bits* que faltarem para se terem os 11 *bits*, assim pode dar-se o caso de não se ter CRC ou ter no máximo 5 *bits*. O bit inicial da mensagem a transmitir indicará se a trama a transmitir é, ou não, superior a 8 *bits*, permitindo distinguir entre o primeiro *byte* e os restantes *bytes* de uma mensagem.

2.8.1.3 Interligação entre blocos e Buffers Implementados

A interligação entre blocos é feita utilizando um protocolo *handshake*. A Figura 2.19 mostra o protocolo entre dois módulos genérico. Para sua implementação são necessárias dois componentes, uma no lado que tem informação a transmitir e outra no lado que recebe informação. Neste capítulo serão referidas duas expressões "protocolo handshake A" e "protocolo handshake B", referindo-se a primeira à parte de transmissão de dados e a segunda à receção.

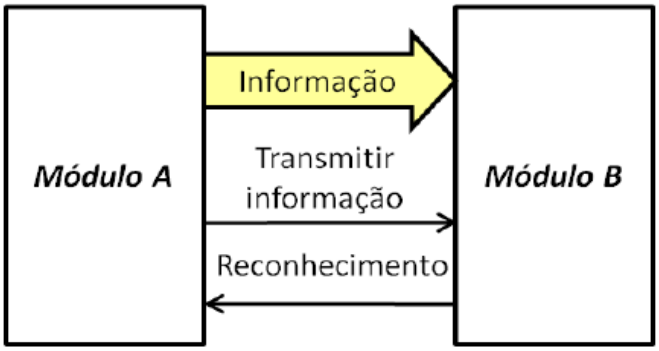


Figura 2.19: Protocolo *handshake* [11]

A implementação foi feita utilizando um modo síncrono, no entanto o ideal seria ter aplicado máquinas assíncronas, não dependentes do sinal de relógio. O facto do *Xilinx Synthesis Technology* (XST) não fazer a síntese de máquina de estado finitas assíncronas [30], fez com que se aplicasse o modo síncrono.

Para a comunicação entre o modelo e os (sub)módulos que interagem diretamente com o modelo, não foi utilizado o protocolo *handshake*, descrito anteriormente, mas sim *buffers* onde será acumulada a informação dos sinais / eventos. A utilização vai permitir que o nó de rede seja transparente aos olhos do modelo.

Os *buffers* implementados são do tipo FIFO e baseados em registos de deslocamento.

2.8.1.4 Receção de tramas série (bloco 1 da Figura 2.16)

Para a receção de tramas através do protocolo série *RS-232*, foi utilizado uma macro *UART*, representada na Figura 2.20. A macro *UART_RX* contém um *buffer* de 16 bytes, ocupa 8 *Configurable Logic Blocks* (CLBs) e fornece o funcionamento de uma *UART* com as seguintes características: *start bit*, oito bits de dados transmitidos em série e o *stop bit*.

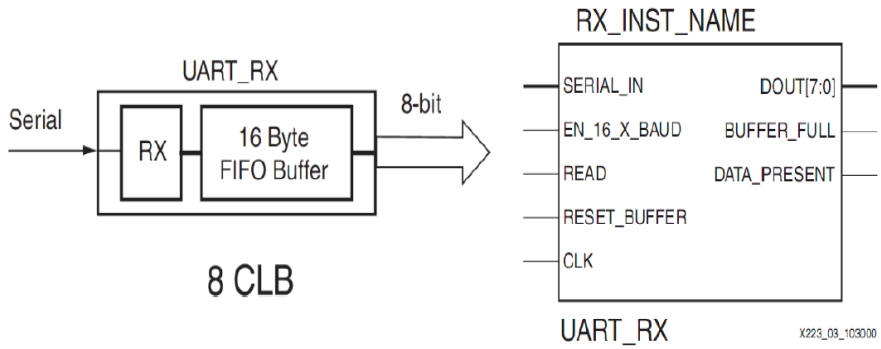


Figura 2.20: Macro de recepção UART [11]

Após a receção do número de bytes necessários, para a construção da mensagem, esta é colocada num *buffer FIFO*. Este *buffer*, que tem como objetivo guardar as mensagens que vão sendo recebidas pelo Nó de Rede, pode ter uma profundidade para oito mensagens com um tamanho de 14 bits (dois bytes recebidos menos os bits de início de mensagem) que equivale ao tamanho da mensagem. O tamanho e a profundidade podem ser alterados consoante as necessidades, ou seja, é possível aumentar ou diminuir o número de mensagens guardadas aumentado ou diminuindo a profundidade do *buffer*. O mesmo acontecendo para o tamanho.

Este bloco tem como sinais de entrada: o sinal de relógio do Nó de Rede, o sinal de *reset* e a linha de comunicação. E como sinal de saída tem as mensagens armazenadas no *buffer*. Além destes, existem mais dois para o protocolo *handshake*, com o bloco da análise da mensagem.

2.8.1.5 Envio de tramas série (bloco 2 da Figura 2.16)

Tal como na receção de dados através do protocolo RS-323, foi utilizada uma macro *UART*, representada na Figura 2.21. A macro *UART_TX* também contém um *buffer* de 16 bytes, ocupa 7 *CLBs* e fornece o funcionamento de uma *UART* com as características descritas para a macro *UART_RX*.

O processo de colocar uma mensagem no *buffer* é igual ao descrito na receção das tramas, mas é feito de forma inversa, isto é, se existir uma mensagem no *buffer* esta é decomposta em N bytes, sendo N o número de bytes necessários para se ter uma mensagem e são colocados na macro *UART_TX*. Este módulo pode receber mais que uma mensagem, uma

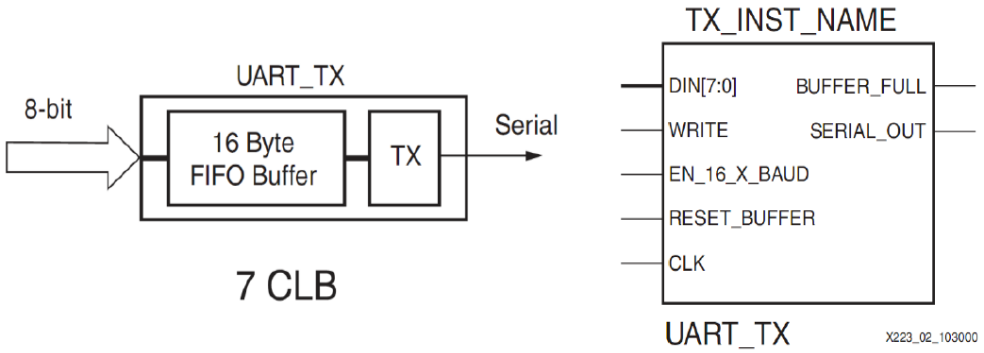


Figura 2.21: Macro de transmissão UART [11]

delas vinda do anel e outras vindas do sistema modelado, provenientes dos módulos inicialmente expressos em RdP. Todos os blocos comunicam entre si utilizando o protocolo *handshake* e é necessário um árbitro que atribuía à vez (uma espécie de testemunho - marca) a cada uma das mensagens para as colocar no *buffer*. Assim, ter-se-á o mesmo número de componentes que implementam o protocolo *handshake* B que o mesmo número de mensagens. O árbitro seleciona qual deverá "processar" a informação.

O *buffer FIFO* utilizado, tem uma profundidade para 8 mensagens, com um tamanho de 14 bits (equivale a dois bytes enviados menos os bits de início da mensagem) que equivale ao tamanho da mensagem. O tamanho e a profundidade podem ser alterados consoante as necessidades, ou seja, é possível aumentar ou diminuir o número de mensagens guardadas aumentando ou diminuindo a profundidade do *buffer*, o mesmo acontecendo para o tamanho.

Este bloco tem como sinais de entrada: o sinal de relógio do Nó de Rede, sinal de *reset*, as mensagens vindas do bloco 3 e do bloco 5. Como sinais de saída, existe a linha de comunicação série e também os sinais para o protocolo *handshake* com blocos 3 e 5.

2.8.1.6 Análise da mensagem recebida (bloco 3 da Figura 2.16)

Antes da análise da mensagem esta é decomposta nos sinais nela contida. Primeiro é verificado através de um comparador se o sinal de destino corresponde ao seu endereço. Se não, coloca a mensagem no buffer de envio de tramas série, descrito na secção anterior. Se igual, através do protocolo *handshake* são enviados os respetivos sinais para o bloco 4 da Figura 2.16, descrito na secção seguinte.

Este módulo tem como sinais de entrada: o sinal de relógio do Nó de Rede, o sinal de *reset* e as mensagens vindas do bloco 1 da Figura 2.16. Como saída, existe o sinal extraído da mensagem e existem também os sinais para o protocolo *handshake* com os blocos referidos.

2.8.1.7 Interface com os sinais/eventos de entrada do módulo (bloco 4 da Figura 2.16)

Este bloco consiste unicamente em aplicar o protocolo *handshake*, mas com uma particularidade. A solução proposta permite que o nó de rede funcione com um sinal de relógio diferente do módulo inicialmente modelado em RdP. O sistema a implementar pode exigir diferentes velocidades entre os vários módulos e o nó deve ser o mais rápido possível para que a transmissão de mensagem custe o menor atraso possível. Para que o Nó de Rede seja "transparente", neste modelo teve que se uma máquina que funcionasse com as velocidades de *clock* do modelo e que implementasse o protocolo *handshake* atrás descrito.

Este bloco funciona com o sinal de relógio do módulo em vez do sinal de relógio do nó, sempre que haja um sinal / evento vindo do bloco 3 da Figura 2.16 explicado anteriormente neste capítulo, e que se queira transmitir esse sinal / evento para o módulo.

O sinais de entrada do bloco são os seguintes: o sinal de relógio do sistema e do módulo, e o sinal de *reset*. Os sinais de saída são os sinais do protocolo *handshake*.

2.8.1.8 Interface com os sinais/eventos de saida do módulo (bloco 5 da Figura 2.16)

Este bloco é composto por duas partes, uma responsável pela interface com o módulo, e a outra, encarregue de transformar o sinal em uma mensagem. A interface tem que permitir captar a informação de forma transparente para o módulo, mesmo que ambos tenham, eventualmente, velocidade de *clocks* diferentes. A utilização do protocolo *handshake* não será possível, pois não é transparente para o módulo e poderia perder-se informação. A solução adotada é a utilização de um sistema GALS, já explicado anteriormente, pois é o que mais se adequa a interfaces assíncronas.

O sinal de relógio do *buffer* é o mesmo do módulo RdP. Sempre que existe um evento ou ocorrer uma variação de sinal é colocado no *buffer*. Caso se trate de um evento, este é diretamente ligado ao *buffer* porque existe o interesse de guardar todos os eventos ocorridos.

A transformação do sinal em mensagens é um processo simples. À partida um sinal terá que ser entregue em um ou mas módulos. Assim sabe-se quantas mensagens terão que ser construídas e quais as condições dos respetivos Nós de Rede e codificações do sinal. Este bloco tem sinais genéricos onde está essa informação.

Este bloco tem como sinais de entrada: o sinal de relógio do Nó de Rede, o sinal de relógio do módulo, o sinal de *reset* e o sinal / evento vindo do módulo. Como sinais de saída: a mensagem que irá para o bloco encarregue de enviar as tramas série (bloco 2) e também os sinais para o protocolo *handshake*.

2.8.2 Ponte

A estrutura da ponte é apresentada na Figura 2.22, onde se pode verificar alterações em relação ao Nó de Rede apresentado na Figura 2.16. Esta estrutura não vai comunicar diretamente com o módulo mas sim com o nó de rede que se encontra situado em outra plataforma.

O funcionamento da Ponte é semelhante ao Nó de Rede. Os blocos 1 e 2 foram explicados na secção 2.6.4 e 2.6.5, respetivamente. Como acontece no Nó de Rede o bloco 1, recebe tramas série vindas de um nó de rede, transformando essas tramas em mensagens. O bloco 2, transforma as mensagens vindas do bloco 3' e do bloco 1 em tramas série e envia essas tramas para um Nó de Rede. A diferença do bloco 2 para o bloco 2', consiste no facto de não ser necessário a utilização do árbitro, basta o protocolo *handshake*, pois só recebe uma mensagem. O bloco 3 difere do bloco 3' pelo facto de, neste último, não ser necessário a descodificação da mensagem para enviar posteriormente os sinais / eventos, basta verificar se o endereço de destino corresponde ao endereço da ponte, se for destinado a este a mensagem será colocada no bloco 2', senão é colocado no bloco 2.

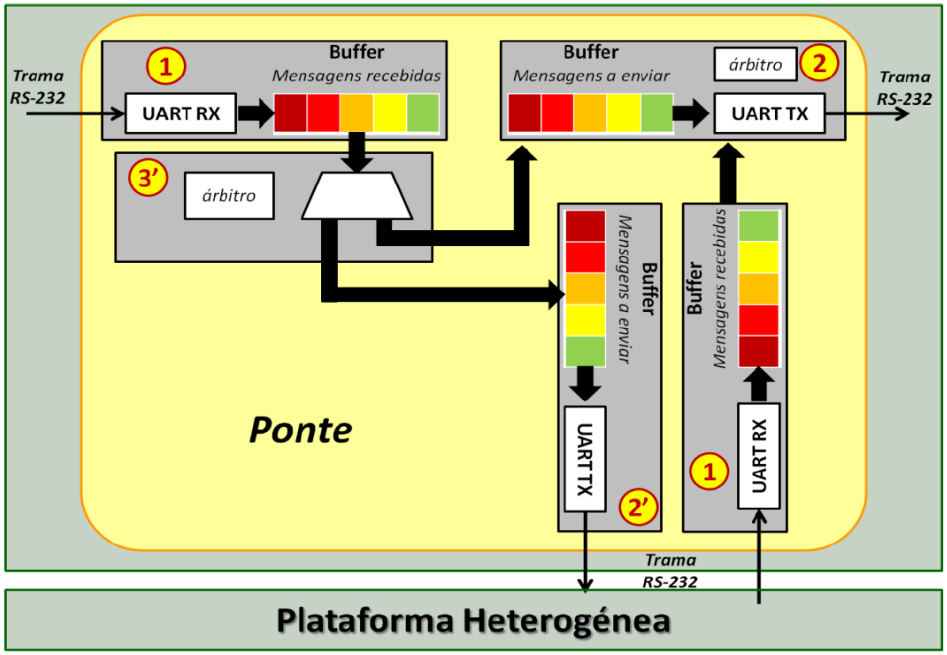


Figura 2.22: Ponte entre plataformas heterogéneas [11]

2.9 Ambiente de desenvolvimento para Redes IOPT

2.9.1 Projeto FORDESIGN

Distintos modelos de computação provaram que se adequam à conceção de sistemas embutidos. Entre estes modelos, encontram-se os diagramas de estado e também as redes de Petri. O projeto FORDESIGN explora a semelhança entre formalismos, tentando, por um lado, agrupá-los numa única metodologia e em ferramentas de desenvolvimento associadas e, por outro lado, propõe a utilização de uma classe de baixo nível de RdP. Esta classe é usada como uma linguagem de especificação, para a qual todos os formalismos são traduzidos antes de serem verificados e implementados [12].

Um conjunto de procedimentos foi formalmente definido, possibilitando a partição do modelo e identificação dos submodelos, reconhecidos como componentes. Cada componente pode ser mapeado em hardware ou software, de acordo com o seu custo específico, utilizando técnicas de co-design hardware-software.

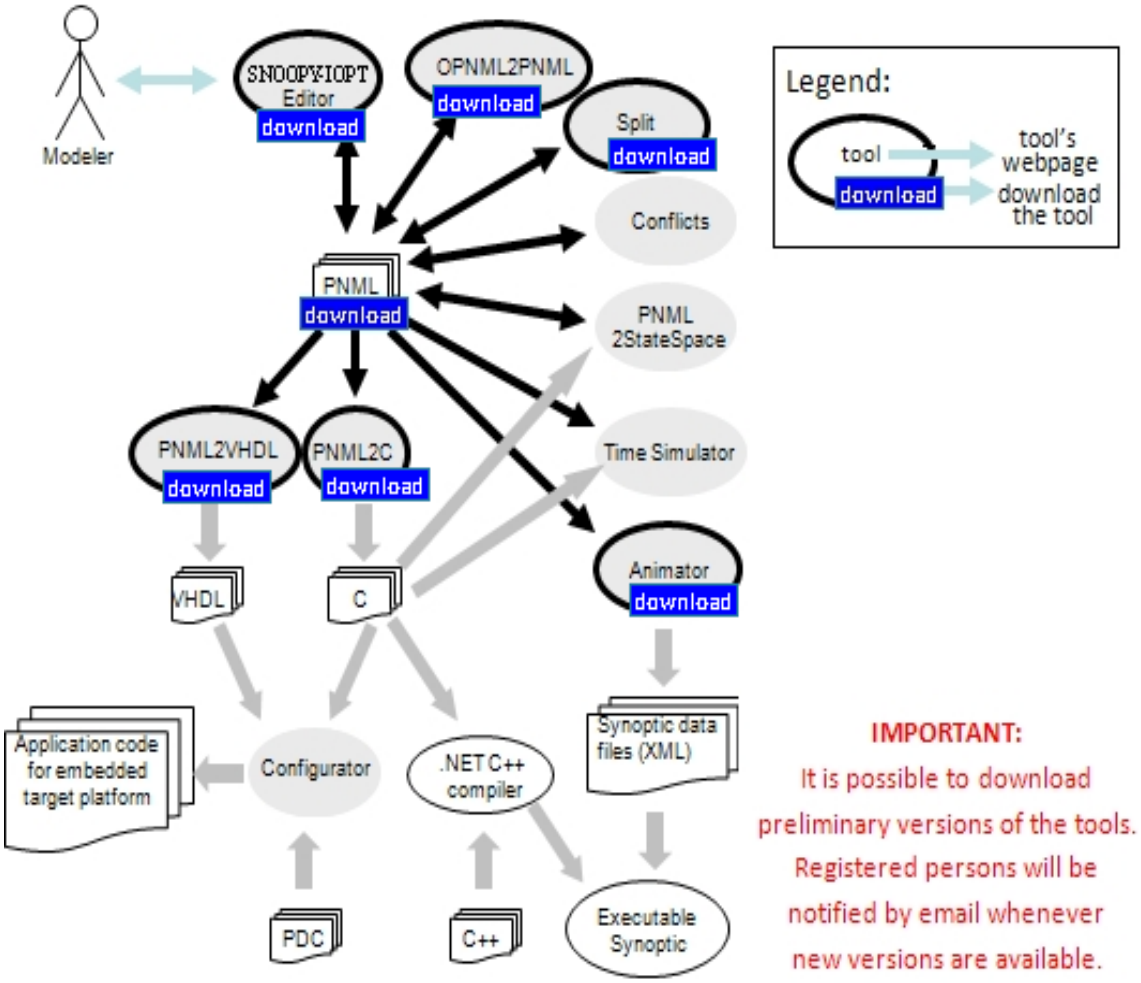


Figura 2.23: Ferramentas em desenvolvimento no projeto FORDESIGN [12]

A Figura 2.23, retirada de [12], mostra as ferramentas que foram desenvolvidas, que se baseiam na representação de RdP através do padrão PNML (já referido anteriormente). As elipses que têm o contorno a preto acentuado, são as ferramentas para as quais já existem versões preliminares.

As ferramentas desenvolvidas são as seguintes:

- Snoopy IOPT – implementa um editor gráfico de RdP para a classe IOPT, suporta hierarquia e especificações modulares e também suporta a representação PNML .
- SPLIT – é uma ferramenta de partição de um modelo IOPT em vários submodelos, usando canais de comunicação síncronos.
- PNML2C – é uma aplicação que gera automaticamente o código ANSI C com origem na representação PNML.
- PNML2VHDL – é uma ferramenta que permite a geração automática de código VHDL a partir da representação PNML de um modelo RdP IOPT.

2.9.2 Ambiente IOPT-Tools

O ambiente IOPT-Tools é um projeto ainda em desenvolvimento que tem como objetivo, a implementação de uma aplicação *Web* para edição de RdP. Este projeto que está a ser desenvolvido por membros do GRES (*R&D Group on Reconfigurable and Embedded Systems*) tem como principais diferenças em relação ao projeto FORDESIGN, o facto de ser uma aplicação *Web* onde todas as ferramentas representadas na Figura 2.24 se encontram juntos na mesma aplicação e também a extensão da classe das redes IOPT, com objetivo de permitir a utilização de sistemas distribuídos como são os sistemas GALS.

A extensão da classe das redes IOPT foi feita para permitir a utilização de diferentes domínios temporais e também de canais assíncronos. Os domínios temporais vão permitir associar cada nó da rede IOPT a um diferente componente, sendo cada componente localmente síncrona. Os canais assíncronos vão ser utilizados para especificar a interação entre os diferentes componentes. Um exemplo utilizando diferentes domínios temporais e canais assíncronos será mostrado no capítulo Exemplos de Aplicação. A figura 2.24 apresenta todas as ferramentas disponíveis no ambiente IOPT-Tools.

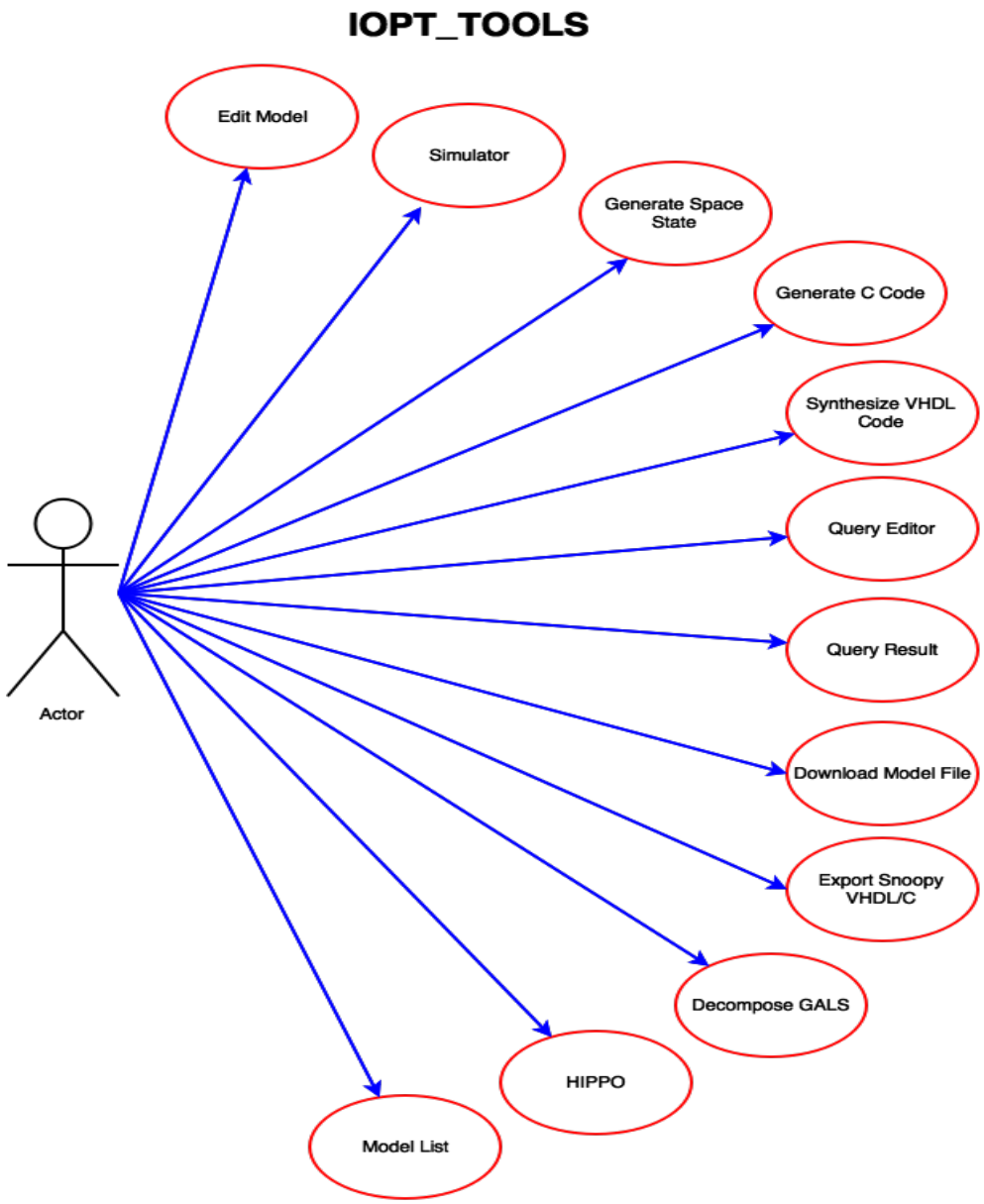


Figura 2.24: Ambiente IOPT-Tools

As ferramentas em desenvolvimento são as seguintes, [17]:

- Edit Model – editor gráfico para as redes de Petri IOPT.
- Simulator – Simulador e *debugger* para as redes de Petri IOPT
- Generate State Space – geração automática do espaço de estados, com origem na representação PNML
- Generate C Code – geração automática de código C com origem na representação PNML, para implementação em micro controladores ou em PC

- Synthesize VHDL Code – geração automática de código VHDL, para implementação em FPGAs ou ASIC
- Query Editor – permite definir uma lista de *queries* que automaticamente valida propriedades durante a execução do espaço de estados
- Query Result – verifica o resultados das *queries* aplicadas durante a a ultima geração do espaço de estados
- Download Model File – *download* do ficheiro do modelo selecionado para o computador do utilizador
- Export Snoopy VHDL/C – Converte o modelo selecionado em formato PNML compatível com o editor de redes de Petri Snoopy. As expressões matemáticas podem ser convertidas em VHDL ou em C
- Decompose GALS – Decompõe o modelo GALS em vários componentes de acordo com os diversos domínios temporais
- HIPPO – Exporta o modelo para a ferramenta HIPPO e calcula a matriz de incidência.
- Model List – Permite gerir e escolher ficheiros no formato PNML.

2.9.3 Espaço de Estados

Neste secção, será abordado a geração do espaço de estados de sistemas GALS através das redes de Petri IOPT, complementadas com canais assíncronos (ACs) e domínios temporais (TDs), apresentado em [19] e [20]. Os canais assíncronos permitem a especificação da interação entre componentes de todo o sistema GALS, enquanto cada componente é especificado através da redes IOPT, tendo cada componente um domínio temporal associado.

Na figura 2.25, é apresentado um exemplo de uma rede de Petri IOPT, complementada com canais assíncronos e domínios temporais. A figura modela dois componentes. Um componente com domínio temporal "1" e outro com domínio temporal "2", comunicando através de dois canais assíncronos "ac.T1.T3" e "ac.T4.T2", em que cada canal tem um domínio temporal específico ("3" e "4").

A comunicação entre componentes GALS é feita através de eventos, em que o evento de saída de uma transição de um componente passará por um canal assíncrono e será o evento de entrada de uma transição de um outro componente. Cada componente GALS tem um domínio temporal associado. A comunicação entre componentes com domínios temporais diferentes apenas é possível através da utilização de canais assíncronos, sendo que cada componente deve ser definido por um domínio temporal.

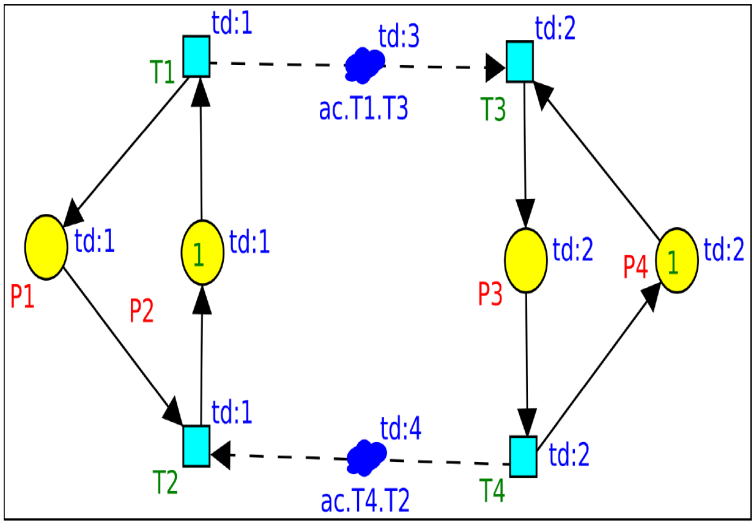


Figura 2.25: Exemplo de uma rede de Petri IOPT com canais assíncronos e domínios temporais

A verificação de sistemas pode ser feita através da análise do espaço de estados. O espaço de estados é um grafo direto com todos os estados que o sistema consegue alcançar. A condição necessária para o sistema evoluir de um estado para outro, é representada na ligação entre os estados (arcos direcionais). A análise ao espaço de estados permite verificar (1) se o sistemas se encontra num estado específico, (2) se um estado indesejado acontece, (3) se o sistema entra em bloqueio (*deadlock*), (4) se através de um estado específico é possível chegar a outro estado específico, entre outros. Através do espaço de estados de sistemas GALS é ainda possível obter informação sobre os recursos de memória necessários à implementação de cada componente e dos canais de comunicação entre os componentes.

Para a obtenção dos espaços de forma automática, foi utilizada a ferramenta IOPTGALS2SS. Esta a ferramenta tem como entrada um ficheiro PNML com um modelo utilizando um sistema GALS especificado através das redes de Petri IOPT, complementadas com ACs e TDs. A saída será a geração do espaço de estados, bem com a seguinte formação adicional:

- número total de nós;
- número total de bloqueio e a sua identificação;
- número total de conflitos e a sua identificação;
- número máximo de *bounds* de cada lugar da rede de Petri IOPT;
- número mínimo de *bounds* de cada lugar da rede de Petri IOPT;
- tamanho máximo dos *buffers* de comunicação associado a cada canal assíncrono;
- tamanho mínimo dos *buffers* de comunicação associado a cada canal assíncrono.

A informação dos recursos de memória necessários para implementar os componentes e os canais assíncronos utilizados nos sistemas GALS, são obtidas através dos últimos quatro *items* apresentados em cima. Um exemplo de um espaço de estados gerado pela ferramenta IOPTGALS2SS é mostrado na figura 2.26.

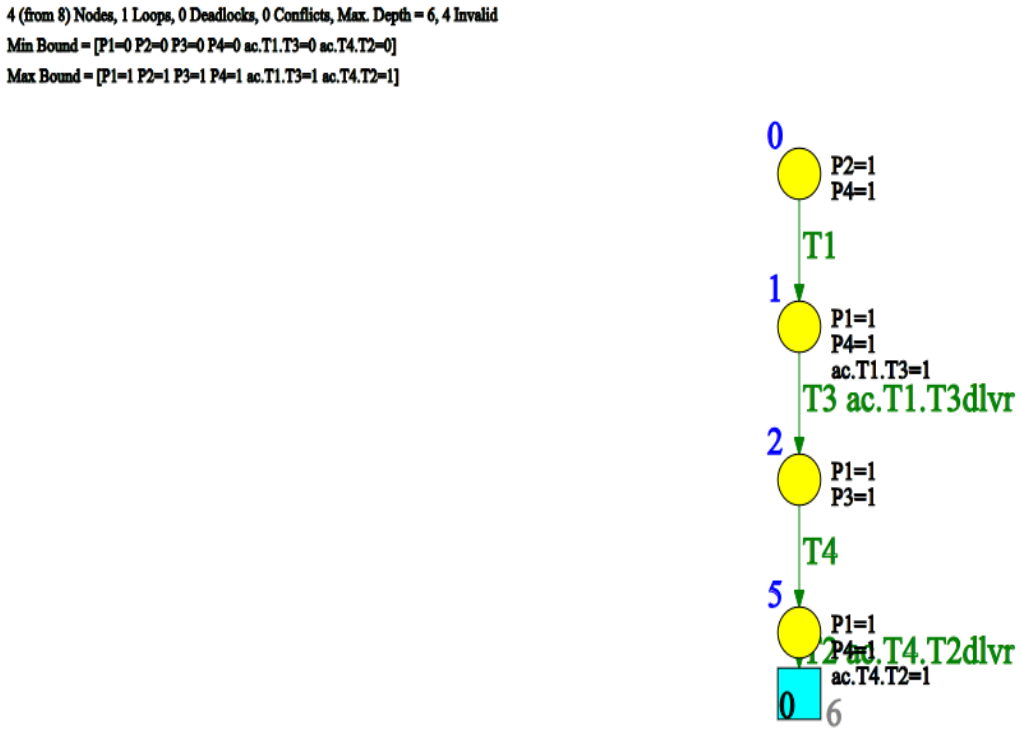


Figura 2.26: Espaço de estados da figura 2.25

3

Soluções para suporte à comunicação

3.1 Topologia Nó de Rede

Nesta secção, serão apresentadas quatro topologias, são elas: ponto a ponto, circular, matriz e toroidal. Cada uma destas topologias permite que cada componente tenha um ou mais nós de rede associados, onde é possível criar, enviar, receber e ler mensagens.

3.1.1 Interligação com série ponto a ponto

Na interligação com série ponto a ponto, os componentes vão comunicar entre si utilizando canais de comunicação entre o componente de origem das mensagens e o de destino. Cada componente pode trocar mensagens com mais do que um componente, para isso basta que esses componentes estejam ligadas através de canais de comunicação. Cada canal de comunicação vai ter dois nós de rede, um associado ao componente de origem de mensagens ("nó de envio") e outro associado ao componente de destino ("nó receção") [22].

A figura 3.1, mostra um exemplo de uma interligação série ponto a ponto, em que o componente "4", que se encontra conectada aos componentes "1" e "3", através dos canais de comunicação representados por arcos na figura 3.1, terá dois nós de envio, por cada canal de comunicação, enquanto os componente "1" e "3" terão respetivamente um nó de receção por cada canal de comunicação com origem no componente "4".

Na figura 3.2, é apresenta uma arquitetura para o nó de envio e uma para o nó de receção.

O nó de envio é constituído por um conjunto de *buffers*, *Buffer evT2* e *Buffer evT3*, para

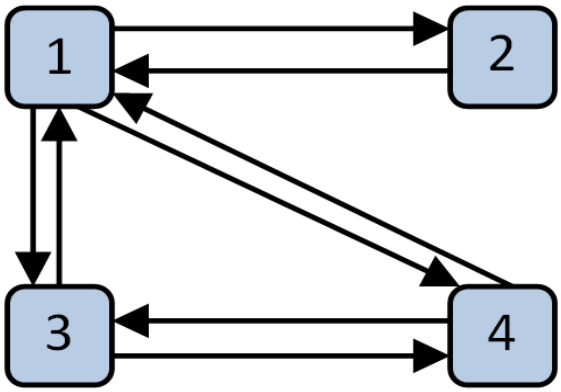


Figura 3.1: Interligação série ponto a ponto

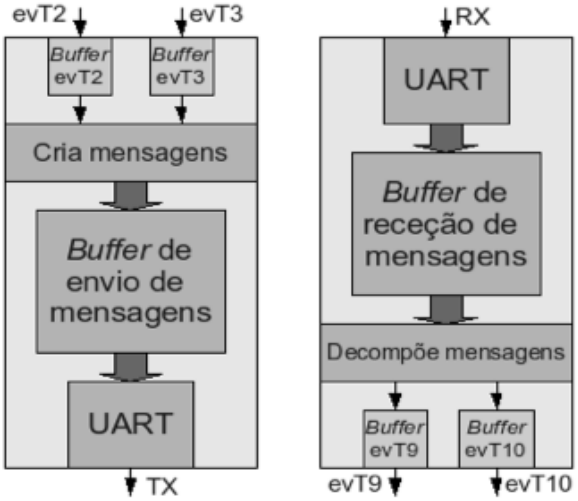


Figura 3.2: Arquitetura para o nó de envio de eventos (lado esquerdo), e de receção de eventos (lado direito) [23]

guardar o número de ocorrências de cada evento (*evT2*, *evT3*) criado pela respetiva transição e também por um *buffer FIFO* para armazenar as mensagens que serão enviadas. Cada mensagem é criada por cada ocorrência de um determinado evento.

O nó de receção, é constituído por um *buffer FIFO*, para armazenar as mensagens recebidas e também por um conjunto de *buffers*, *Buffer evT9* e *Buffer evT10*, para armazenar o número de ocorrências de cada evento (*evT2*, *evT3*).

Na interligação ponto a ponto a troca de mensagens entre componentes é sempre feita pelo caminho mais curto, as mensagens não precisam do endereço de origem e destino, como acontece com as outras interligações explicadas a seguir. Em comparação com os outros tipos de interligações explicadas nesta secção, a comunicação é mais rápida o *overhead* é menor e os nós de rede utilizados são menos complexos. A desvantagem desta

interligação é o número de canais de comunicação e o número de nós de rede utilizados, que são em maior número do que nas outras interligações.

3.1.2 Interligação em barramento

A interligação bus permite que cada componente comunique entre si, utilizando um barramento de comunicação. Cada componente vai enviar e receber mensagens através do Bus de comunicação. A figura 3.3 mostra um exemplo de uma interligação Bus com 5 componentes em que a cada componente está associado um nó de rede.

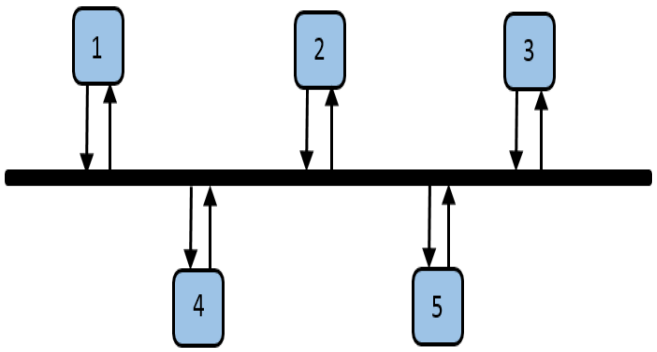


Figura 3.3: Interligação em bus

O componente 1 da figura 3.3, para comunicar com o componente 5 vai enviar a mensagem através do Bus. Apenas o componente de destino vai aceitar a mensagem, pois a mensagem contém a informação com o componente que enviou e com o componente de destino. Só assim é possível que os componentes de destino saibam que a mensagem se destina a eles [22].

A desvantagem desta interligação é o *overload* de mensagens no barramento, caso hajam muitas mensagem a circular no barramento a comunicação é mais lenta.

3.1.3 Interligação em série circular

A interligação com série circular permite que cada componente tenha apenas um nó de rede associado. Cada componente recebe mensagens sempre do componente anterior a ela e envia mensagens sempre para o componente seguinte. A figura 3.4, mostra um exemplo de uma interligação com série circular com 4 componentes em que a cada componente está associado um nó de rede [22].

O componente "1", da figura 3.4, vai receber sempre mensagens através do componente "4" e enviar mensagens sempre para o componente "2". Caso uma mensagem recebida, seja para outro componente, ela será enviada para o componente seguinte, até encontrar o destino certo. Por exemplo quando o componente "1" recebe uma mensagem destina

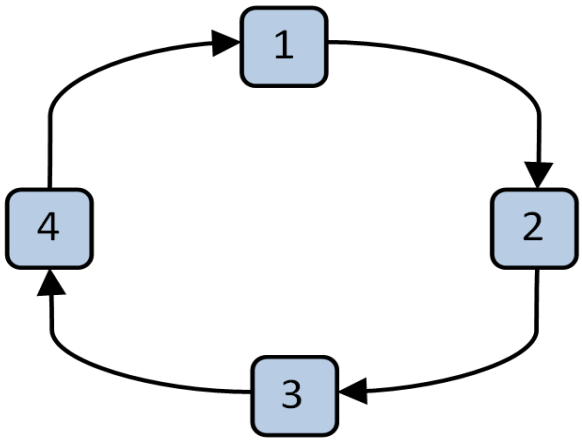


Figura 3.4: Interligação série circular

ao componente "3", essa mensagem será enviada para o componente "2", que enviará posteriormente para o componente "3".

Esta interligação tem como vantagens o facto de ser necessário apenas um nó de rede por componente e a simplicidade da rede, que recebe e envia mensagens sempre através dos mesmos componentes. As desvantagens associadas a esta topologia são: o grande número de mensagens a circular na rede, o tempo que uma mensagem demora a percorrer todas os componentes até atingir o destino e também, caso uma das ligações entre componentes falhe, toda a rede ou uma parte dela irá falhar.

3.1.4 Interligação circular dupla

A interligação circular dupla permite que cada componente tenha apenas um nó de rede associado. Cada componente recebe e envia mensagens da componente anterior e da seguinte. A figura 3.5 mostra um exemplo de uma interligação com série circular dupla com 4 componentes em que a cada componente está associado um nó de rede.

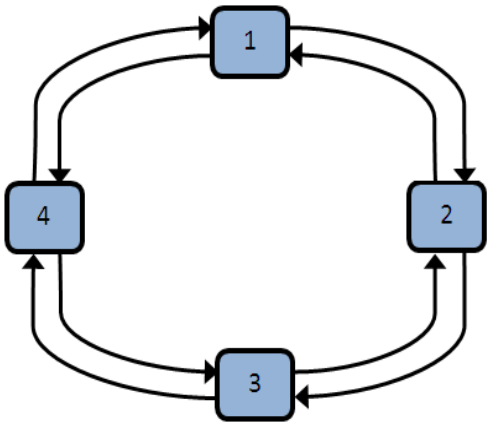


Figura 3.5: Interligação circular dupla

A interligação circular dupla tem dois sentidos de comunicação entre os componentes, um no sentido dos ponteiros do relógio e outro no sentido contrário. O facto de se poder comunicar em qualquer um dos sentidos é uma vantagem em relação à interligação série circular, onde só é permitido circular em apenas um dos sentidos. Caso a comunicação entre componentes falhe num dos sentidos, é sempre possível essa comunicação se manter, pois existe sempre o sentido contrário para se efetuar a comunicação. Outra das vantagens de se comunicar nos dois sentidos é o tempo que uma mensagem demora a chegar à componente de destino, pois é possível verificar-se qual o caminho mais curto para enviar as mensagens de comunicação.

A desvantagem associada a esta interligação é a complexidade dos nós de rede, porque cada nó permite enviar e receber mensagens do componente anterior e também do componente seguinte.

3.1.5 Interligação em matriz

Na interligação com matriz, os componentes são bidirecionais, isto significa que cada componente tanto pode enviar mensagens como receber mensagens. Cada componente dependendo da sua posição na matriz, vai comunicar no mínimo com dois outros componentes e no máximo com 4.

O caso do componente 1 da figura 3.6, que é a primeira componente da matriz vai comunicar apenas com dois componentes, os componente 2 e 4, isto é o número mínimo de ligações. O número máximo de ligações pode ser verificado no componente 5, que comunica com os componentes 2, 4, 6 e 8.

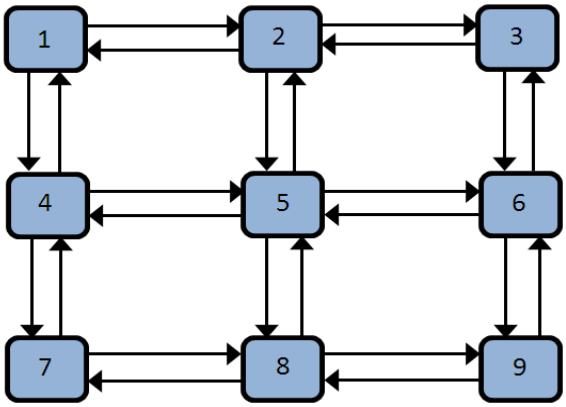


Figura 3.6: Interligação em matriz

Esta interligação permite escolher o caminho mais curto para comunicação entre componentes e também escolher caminhos alternativos caso essas ligações de comunicação estejam congestionadas com muitas mensagens. Uma desvantagem é a complexidade de

cada componente, pois esta complexidade vai depender da posição da componente na matriz.

3.1.6 Interligação toroidal

Na interligação toroidal, os componentes da última linha e da ultima coluna vão se ligar aos componentes da primeira linha e da primeira coluna, respetivamente. A figura 3.7 mostra um exemplo de uma interligação toroidal com 6 componentes e cada componente está associado um nó de rede.

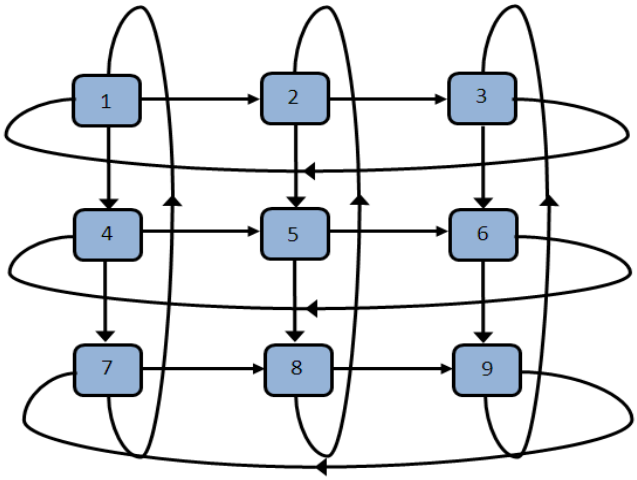


Figura 3.7: Interligação em toroidal

Cada componente vai receber informação do componente localizada na coluna anterior e vai enviar informação para o componente na coluna seguinte (mesma linha) e também para o componente da linha seguinte (mesma coluna). As exceções são a primeira linha e coluna e também a última linha e coluna. As primeiras linha e coluna vão receber informação da última linha e coluna, respetivamente. A última linha e última coluna foram explicadas no parágrafo anterior.

A vantagem desta interligação é do último componente (linha/coluna) se ligar com o primeiro o que permite que a comunicação seja feita mais rapidamente já que a informação não vai passar por componentes intermédias. Uma desvantagem é facto de um componente não poder trocar informação diretamente com o componente anterior. Por exemplo: a comunicação do componente 4 com o componente 1 da figura 3.7, só pode ser feita através do componente 6, o que faz com o tempo de comunicação entre estes componentes seja maior.

4

Sistema Desenvolvido

Este capítulo tem como objetivo apresentar o protótipo desenvolvido, focando-se no ambiente de desenvolvimento, nos cenários contemplados e nos ficheiros produzidos.

Tal como foi mencionado no início deste documento, o principal objetivo desta dissertação é apresentar uma ferramenta que permita a geração automática de um ficheiro de configuração para diversas plataformas. Devido ao facto de haver poucas ferramentas que permitem a geração de ficheiros de configuração para diversas plataformas, a solução apresentada, tem como objetivo resolver este problema.

4.1 Enquadramento da Solução

Tendo em conta que o objetivo desta ferramenta é ser integrada no ambiente IOPT-Tools, que foi explicado com detalhe no Capítulo 2, a melhor opção seria o desenvolvimento da ferramenta através da *Web*, permitindo que a ferramenta estivesse sempre *online* para que a integração com o ambiente IOPT-Tools se fizesse da maneira mais simples possível.

Anteriormente a este trabalho, foi desenvolvida uma ferramenta que permite a geração automática de ficheiros de configuração para diferentes plataformas, mas apenas permitia a ligação entre módulos que tivessem uma ligação direta entre si, o que não permitia que existissem módulos intermédios entre os módulos de comunicação. Outra limitações da ferramenta antiga prende-se com o facto de todos os módulos poderem apenas ser implementados numa plataforma, enquanto no protótipo aqui apresentado é possível que cada módulo se encontre numa plataforma diferente.

O protótipo desenvolvido teve como objetivo a resolução destes problemas, ou seja a comunicação dos módulos através de diferentes níveis (não apenas diretamente) e também proporcionar maior flexibilidade de implementação dos módulos em diferentes plataformas.

4.2 Ambiente de Desenvolvimento

Como foi dito anteriormente, a melhor opção seria o desenvolvimento da ferramenta através da *Web*. Para a criação dessa ferramenta, foi utilizada a linguagem HTML, que é uma linguagem que permite a criação de *websites*. A definição de HTML é *HyperText Markup Language*, em que *HyperText* é o método que permite a navegação para outros *websites*, o que significa que, ao carregar nos chamados *hyperlinks* irá carregar um novo *website*. O facto de ser *hyper* significa que não é linear, ou seja, ao se carregar nos chamados *hyperlinks* é possível ir para a outro *website* ou então ir para outra local no mesmo *website*. A utilização de *tags* no HTML serve para mostrar que tipo de informação se quer exibir no *website*, a isto chama-se *Markup*.

Para definir o *layout* do *website* foi utilizado CSS (Cascading Style Sheet). O CSS é uma linguagem para estilos que define o layout de documentos HTML. Por exemplo, o CSS controla fontes, cores, imagens, linhas, alturas, larguras, posicionamentos entre outras coisas. É possível utilizar o HTML para definir o *layout* de páginas Web, contudo o CSS proporciona mais opções, é mais preciso e sofisticado. A diferença entre o HTML e o CSS é a seguinte: o HTML é usado para estruturar conteúdos, enquanto o CSS é utilizado para formatar conteúdos estruturados.

Para adicionar elementos dinâmicos e interativos no *website* foi utilizado javascript e também *jquery* que é uma das bibliotecas mais conhecidas de javascript. O javascript é uma linguagem de programação muito utilizada no desenvolvido de *websites*. É atualmente a principal linguagem para programação *client-side*, isto significa que o código fonte é processado pelo browser em vez do *webserver*, o que permite que as funções em javascript possam ser executadas depois da *webpage* ter sido carregada sem ser necessária a comunicação com o servidor.

O PHP foi outra das linguagens de programação utilizadas, no desenvolvimento da ferramenta, mais concretamente para a geração do ficheiro XML. Ao contrário do javascript, explicado no parágrafo anterior, é uma linguagem *server-side*.

4.3 Sequência de Procedimentos da Ferramenta

A sequência de procedimentos apresentada na figura 4.1, que será aplicada no exemplo apresentado no capítulo 5, é a seguinte, começar por modelar uma RdP-IOPT na aplicação *IOPT-Tools* gerando o ficheiro PNML associado, aplicar a esse mesmo ficheiro

a funcionalidade *Decompose GALS* pertencente à mesma ferramenta, permitindo obter os ficheiros PNML dos submodelos resultantes da decomposição desse ficheiro PNML. Seguidamente cria-se o ficheiro de interligação dos componentes e toda a configuração desejada e por último é gerado o ficheiro com toda a configuração efetuada.

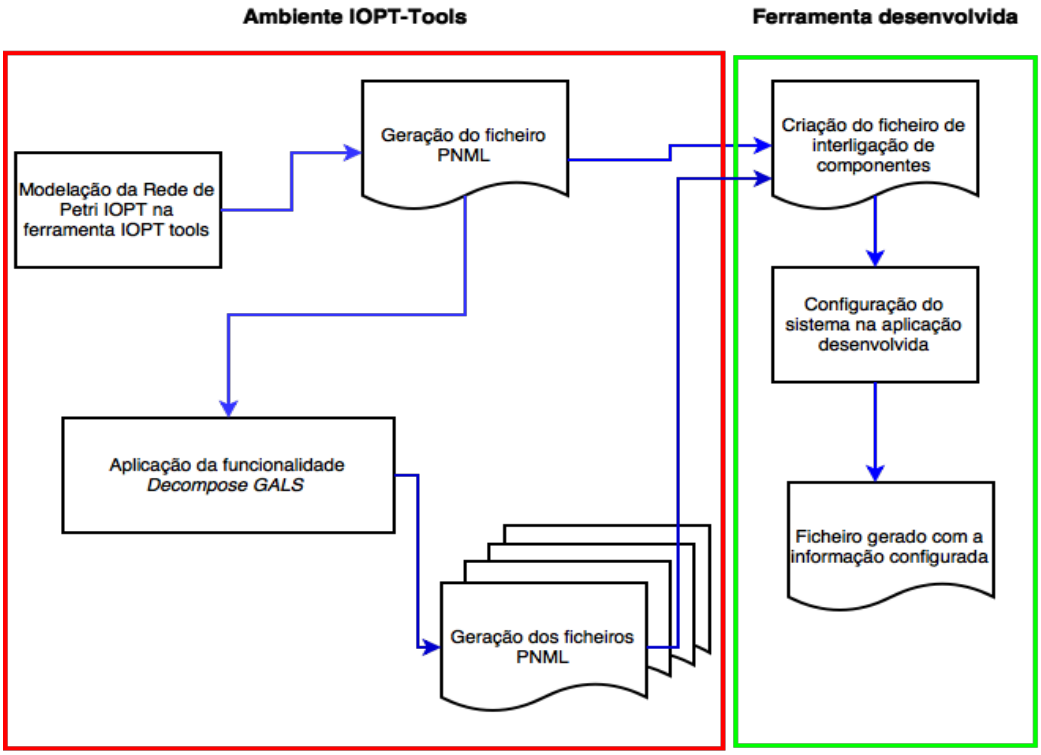


Figura 4.1: Sequência de Procedimentos para geração do exemplo

4.4 Ferramenta Desenvolvida

4.4.1 Casos de Uso

Nesta secção, serão apresentados os casos de uso que foram identificados para melhor descrever a ferramenta que foi desenvolvida. Os casos de uso são apresentados na figura 4.2.

4.4.1.1 Caso de Uso: Escolher Ficheiro

Este caso de uso específica a ação de escolha do ficheiro que vai conter a seguinte informação, os eventos de entrada e saída e os seus respetivos componentes. A estrutura e criação desse ficheiro foram explicadas na secção 4.6.

1. **Actores** - Utilizador

2. **Pré-Condição** - O ficheiro escolhido deve conter informação.

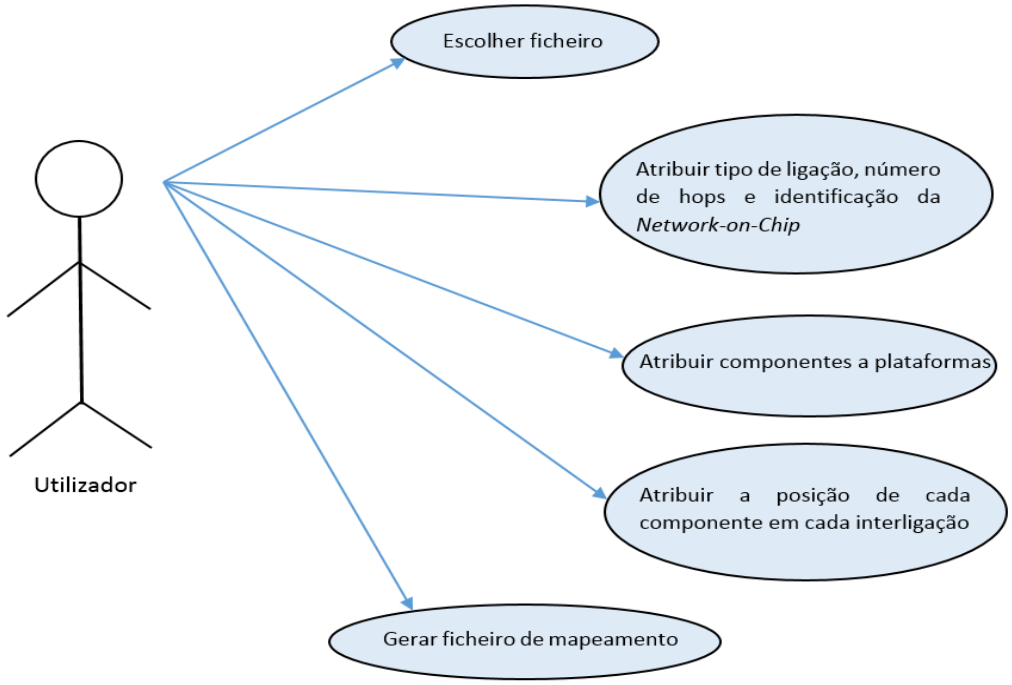


Figura 4.2: Casos de Uso

3. **Fluxo de Eventos**

   (a) Escolher o ficheiro referente ao modelo que se deseja implementar.

   (b) Ferramenta apresenta os componentes de entrada e saída e os respetivos eventos.

4. **Pós-Condição**

   (a) **Condição de Sucesso** - Caso o ficheiro seja lido com sucesso, são apresentados os componentes e respetivos eventos. O utilizador pode continuar para o passo seguinte.

   (b) **Condição de Insucesso** - Se o ficheiro escolhido não tiver o formato correto, será apresentada uma mensagem de erro.

4.4.1.2 Caso de Uso: Atribuir tipo de ligação, número de hops e da identificação da *Network-on-Chip*

Este caso de uso é usado para especificar as ações de escolha do tipo de ligação, número de *hops* e da identificação da *Network-on-Chip*. O número de *hops* representa o número de componentes pelo qual um evento irá passar, nesse número está incluído o componente origem e o de destino. O tipo de ligação já foi explicado nas secções 2.7 e 3.1. A identificação da *Network-on-Chip* será necessária caso sejam escolhidos os seguintes tipos de ligação, *Bus*, *Ring*, *Double Ring*, *Matrix* e Toroidal, que são especificadas no capítulo

três na secção 3.1. Essa identificação é necessário para a utilização de por exemplo, dois *Rings* distintos em que é necessário identificar cada um deles, para depois se atribuir os componentes aos respetivos *Rings*.

1. **Actores** - Utilizador
2. **Pré-Condição** - A ferramenta apresentar os componentes e os respetivos eventos.
3. **Fluxo de Eventos**
   (a) Escolher o número de hops (*Nr of Hops*).
   (b) Escolher o tipo de ligação (*Type Connections*) que liga o evento de saída ao evento de entrada.
   (c) Dependendo do tipo ligação escolhida, é atribuída uma identificação (*Id Network-On-Chip*), sendo que essa identificação é um valor numérico.
4. **Pós-Condição**
   (a) **Condição de Sucesso** - Depois de escolhido o número de *hops*, o tipo de ligação e caso necessário a sua identificação, deverá salvar-se essa informação através do botão de "*Save*".
   (b) **Condição de Insucesso** - Caso não seja introduzida nenhum tipo de ligação irá ser apresentado um erro.

4.4.1.3 Caso de Uso: Atribuir componentes a plataformas

Este caso de uso específica a ação de escolher a plataforma a atribuir a cada componente.

1. **Actores** - Utilizador
2. **Pré-Condição** - Haver pelo menos um componente de saída e entrada com tipo de ligação escolhida e essa informação ter sido salva através do botão de "*Save*".
3. **Fluxo de Eventos**
   (a) Escolher uma identificação para a plataforma (*Id Platform*).
   (b) Escolher o tipo de plataforma (*Type of Platform*).
   (c) Escolher qual o tipo de implementação, que vai estar dependente da plataforma escolhida.
   (d) Escolher o *Time Domain* que pretende para a plataforma especificada anteriormente.
   (e) Escolher o módulo que vai ser implementado na plataforma escolhida.
   (f) Adicionar toda a informação escolhida através do botão "*Add to List*".
4. **Fluxo Alternativo de Eventos**

(a) **Inválido *"Id Platform"*** Ao se adicionar um "*Id Platform*" que já tenha sido atribuído a uma outra plataforma, então

i. será apresentada uma mensagem de erro.

ii. fechando o erro, é possível corrigir o problema e adicionar novamente.

(b) **Inválido *"Time Domain"*** Ao se adicionar um diferente "*Time Domain*" a uma plataforma com uma identificação de plataforma e um *Time Domain* já associados, então

i. será apresentada uma mensagem de erro

ii. fechando o erro, é possível corrigir o problema e adicionar novamente

(c) **Inválido *"Modulo"*** Ao se adicionar um "*Modulo*" que já tenha sido adicionado anteriormente, então

i. será apresentada uma mensagem de erro

ii. fechando o erro, é possível corrigir o problema e adicionar novamente

5. **Pós-Condição**

(a) **Condição de Sucesso** - Caso toda a informação tenha sido inserida sem problemas, deverá salvar-se essa informação através do botão de "*Save*".

(b) **Condição de Insucesso** - Se nenhuma informação foi adicionada a lista através do botão "*Add to List*", ao se tentar salvar irá surgir uma mensagem de erro.

4.4.1.4 Caso de Uso: Atribuir a cada interligação a informação pretendida

Este caso de uso específica a ação de atribuir a posição de cada componente em cada interligação, sendo essas interligações as seguintes:*Serial*, *Bus*, *Ring*, *Double Ring*, *Matrix* e *Toroidal*.

1. **Actores** - Utilizador

2. **Pré-Condição** - A escolha do tipo de ligação ser uma das seguinte opções: *Serial Bus*, *Ring*, *Double Ring*, *Matrix* e Toroidal.

3. **Fluxo de Eventos**

(a) Escolher o tipo para cada uma das ligações

(b) Para o caso de *Serial* escolher o *Baud Rate* e também *Stop Bit*.

(c) Escolher a posição do módulo na interligação para os casos de *Bus*, *Ring*, *Double Ring*, *Matrix* e *Toroidal*

(d) Salvar a informação introduzida, através do botão de "*Save*".

4. **Pós-Condição**

(a) **Condição de Sucesso** - Todas os componentes da mesma interligação terem posições distintas.

(b) **Condição de Insucesso** - Se houver componentes da mesma interligação na mesma posição, será apresentado uma mensagem de erro.

4.4.1.5 Caso de Uso: Gerar ficheiro de mapeamento

O caso de uso aqui especificado tem como ação gerar o ficheiro que vai conter toda a informação referente ao mapeamento feito nos casos de uso anteriores.

1. **Actores** - Utilizador
2. **Fluxo de Eventos**
   (a) Gerar o ficheiro através do botão "*Generate Files*"
3. **Pós-Condição**
   (a) **Condição de Sucesso** - Ficheiro Gerado com sucesso

4.5 Wrapper

4.5.1 FIFO Buffer

A composição da *FIFO Buffer* varia consoante o seu dimensionamento, esse dimensionamento é obtido através da geração do espaço de estados, utilizando a ferramenta desenvolvida no ambiente IOPT-Tools, que implementa o algoritmo proposto em [19]. Nas Figuras 4.3, 2.14 e 4.4 são apresentadas as *FIFOs* com *buffers* de dimensão 1, 2 e 3 respetivamente.

O *semi-decoupled latch controller* (SDLC) e o *latch* fazem parte da composição da *FIFO*. O número de *SDLCs* e de *latchs* vai depender do dimensionamento escolhido, como se pode ver na Figura 4.3, 2.14 e 4.4, assim sendo, caso a dimensão do *buffer* seja de 1, a *FIFO* é constituída por um SDLC e por um *latch*, caso o *buffer* seja de dimensão 2 a *FIFO* é constituída por dois SDLC e por dois *latchs* e assim sucessivamente.

4.5.2 Ponte Wrapper

Na Figura 4.5 estão representadas duas estruturas, sendo a primeira constituída pelo *Input Port* e pela *FIFO* e a segunda pelo *Output Port*. Essas duas estruturas que resultaram da partição do *wrapper assíncrono* descrito na secção 2.4 do capítulo 2, vão permitir que diferentes submodelos, obtidos através da partição de um modelo inicial através da ferramenta SPLIT desenvolvida no âmbito do projeto FORDESIGN, possam comunicar entre si, estando estes submodelos em plataformas distintas.

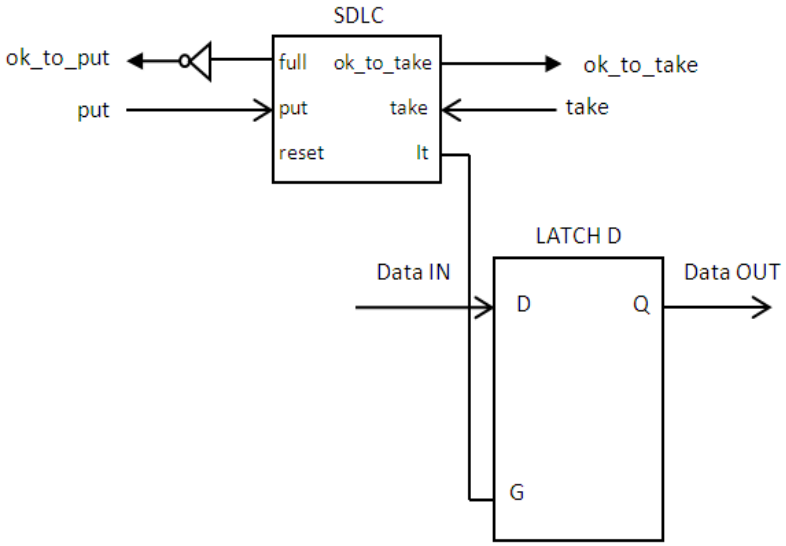


Figura 4.3: *FIFO* do *wrapper assíncrono* com dimensão 1

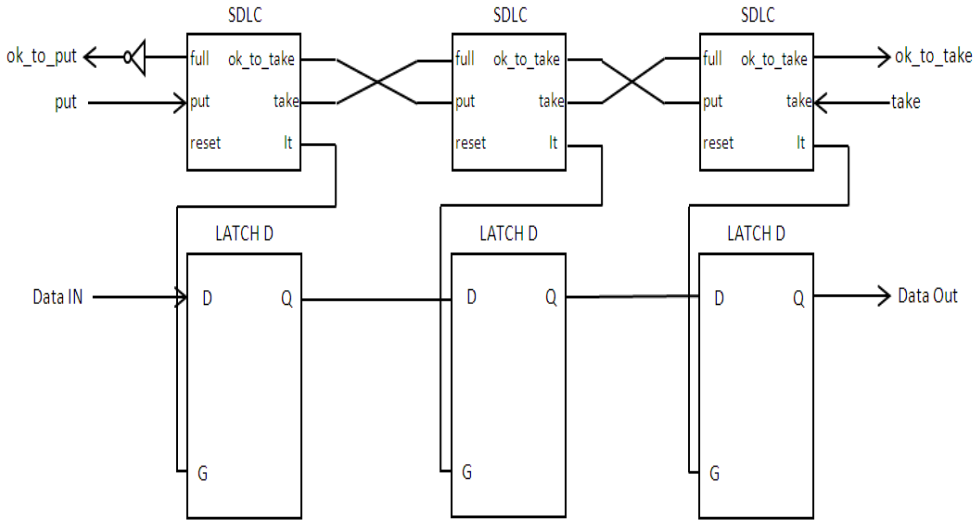


Figura 4.4: *FIFO* do *wrapper assíncrono* com dimensão 3

A comunicação entre submodelos, que utilizem a mesma plataforma, é feita utilizando *wrapper assíncrono*, entre submodelos situados em plataformas distintas foi criado o ponte *wrapper*. O seu funcionamento é igual ao do *wrapper assíncrono*, a única alteração feita, consiste na separação do *Output Port* da *FIFO* e do *Input Port* como se pode ver na Figura 4.5.

A interligação entre os diferentes submodelos utilizando a ponte *wrapper*, é feita da seguinte forma: num das plataformas vai estar a parte do *Input Port* com a *FIFO*, que se iriam ligar na outra plataforma ao *Output Port*, permitindo assim a comunicação GALS entre componentes em diferentes plataformas.

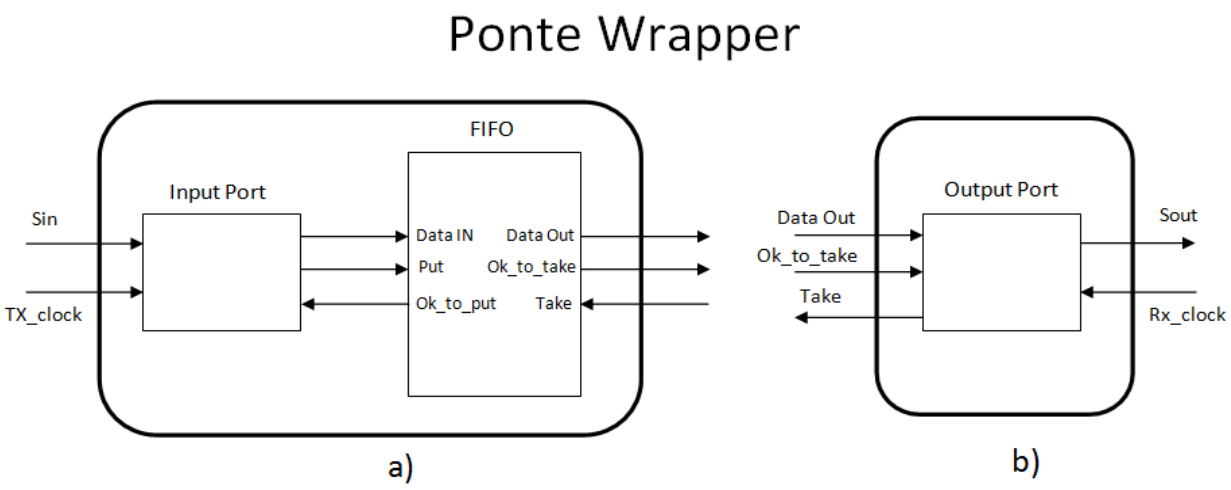


Figura 4.5: A) Estrutura do *Input Port*, com a *FIFO*, B) *Output Port*

4.6 Ficheiro de interligação entre os componentes

Esta secção apresenta o formato do ficheiro criado através de informação recolhida do ficheiro PNML do modelo e também dos ficheiros PNML obtidos através da separação do ficheiro PNML do modelo. A estrutura do ficheiro é apresentada na figura 4.7.

```
<?xml version="1.0" encoding="UTF-8"?>
<interconnection>
  <distributedmodel filename="" netname=""/>
  <submodels>
    <submodel submodelid="" filename="" netname=""/>
  </submodels>
  <asynchronouschannel acid="" acname="" bound="">
    <outputevent submodelid="" eventid=""></outputevent>
    <inputevent submodelid="" eventid=""></inputevent>
  </asynchronouschannel>
</interconnection>
```

Figura 4.6: Estrutura do ficheiro de interligação entre os componentes

Para a criação deste ficheiro foi definido um formato INT, que é um formato XML, apenas tem este nome para se poder distinguir com facilidade do ficheiro XML gerado pela ferramenta, que é explicado no capítulo seguinte.

No ficheiro estão presentes os seguintes componentes:

- **distributedmodel** - É constituído pelos seguinte atributos: *filename* e *netname*. O atributo *filename* é o nome do PNML do modelo, enquanto o *netname* vai ter o mesmo nome a exceção é a não inclusão do formato do ficheiro, que é incluído no primeiro atributo.
- **submodels** - A informação relativa a todos os submodelos gerados pela separação do ficheiro PNML do modelo encontra-se neste elemento;
- **submodel** - Os atributos que fazem parte deste elemento são os seguintes: *submodelid*, *filename* e *netname*. O atributo *submodelid* é um identificar numérico que permite identificar facilmente o submodelo pretendido, *filename* é o nome do PNML do modelo e o formato do ficheiro utilizado, enquanto o *netname* tem o mesmo nome do *filename*, a exceção, é a não inclusão do formato do ficheiro;
- **asynchronouschannel** - Cada canal assíncrono representado no ficheiro PNML do modelo é apresentado através deste elemento. Os atributos constituintes deste elemento são os seguintes, *acid*, *acname* e *bound*. O atributo *acid* é um identificador numérico que permite identificar do canal assíncrono utilizado, *acname* é nome do canal assíncrono, enquanto o *bound* é um valor numérico que representa o número de marcas que o lugar associado à transação vai ter, sendo que essa transição está ligada ao canal assíncrono;
- **outputevent** - Este elemento vai conter a informação relativa ao evento de saída do submodelo, esse evento de saída é identificado através do atributo *eventid* e o modelo através do atributo *submodelid*;
- **inputevent** - Este elemento vai conter a informação relativa ao evento de entrada no submodelo, esse evento de entrada é identificado através do atributo *eventid* e o modelo através do atributo *submodelid*.

4.7 Ficheiro Gerado

Para se poder gerar o código de implementação para diversas plataformas sendo essas plataformas as seguintes: Spartan3 Starter Kit Board, BASYS2, Kintex7, Virtex7, ZedBoard(ZYNQ), Arduino Duemilanove, Arduino Uno, Arduino Duo e Pic 18F4620, das plataformas referidas são mostradas no apêndice A. Foi gerado um ficheiro onde vai estar a informação relativa a todos os dados que são introduzidos pelo utilizador na ferramenta. O ficheiro é gerado em formato XML, porque é um formato que tanto é compreendido por qualquer pessoa, como também é facilmente interpretada por qualquer linguagem de programação.

O ficheiro XML tem o formato apresentado na figura 4.7. Na figura 4.7 é possível identificar as diferentes opções para suporte a comunicação que foram explicadas no capítulo 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configurator>
  <Events>
    <Event nrHops="" hops="" outputmodulo="" outputevent="" inputevent="" inputmodulo="" bound="" typeconnection="" idnetwork=""/>
  </Events>
  <Platforms>
    <Platform id="" typeplatform="" typeimplementation="" timedomain="" modulo=""/>
  </Platforms>
  <Wrappers>
    <Wrapper outputmodulo="" inputevent="" outputevent="" inputmodulo=""/>
  </Wrappers>
  <Serials>
    <Serial outputmodulo="" outputevent="" inputevent="" inputmodulo="" type="" id=""/>
    <Rate serialid="" baudrate="" stopbit=""/>
  </Serials>
  <Buses>
    <Networkbus id="" busnetworktype="">
      <Bus modulo="" position=""/>
    </Networkbus>
  </Buses>
  <Rings>
    <Networkring id="" ringnetworktype="">
      <Ring modulo="" position=""/>
    </Networkring>
  </Rings>
  <DoubleRings>
    <Networkdoublering id="" douringnetworktype="">
      <DoubleRing modulo="" position=""/>
    </Networkdoublering>
  </DoubleRings>
  <Matrixs>
    <Networkmatrix id="" matrixnetworktype="">
      <Matrix modulo="" row="" column=""/>
    </Networkmatrix>
  </Matrixs>
  <Toroidals>
    <Networktoroidal id="" toroidalnetworktype="">
      <Toroidal modulo="" row="" column=""/>
    </Networktoroidal>
  </Toroidals>
</Configurator>
```

Figura 4.7: Estrutura genérica do ficheiro XML

Os principais componentes presentes no ficheiro XML são os seguintes:

- **Events** - Contém toda a informação referente a todos os eventos que são listados pela ferramenta. Para cada evento são listados os seguintes campos: *nrHops* que permite ao utilizador escolher quantos módulos se vão ligar entre si, *hops* mostra se os módulos se encontram ligados diretamente(Hop) ou indiretamente(MultiHop),

outputmodulo representa o módulo de saída, *outputevent* mostra o evento de saída, *inputevent* representa o evento de entrada , *inputmodulo* mostra o módulo de saída, *bound* já foi explicado anteriormente no capítulo 2 na seção 2.9.3, *typeconnection* mostra o tipo de ligações que o utilizador pode escolher e *idnetwork* que só é mostrado para alguns tipos de ligação (*Bus*, *Ring*, *Double Ring*, *Matrix* e Toroidal), permite identificar cada uma das ligações atribuídas.

- **Wrappers** - Contém a informação de todos os *wrappers* em que cada *wrapper* vai ter a seguinte informação, um módulo de saída, um módulo de entrada, um evento de saída e um evento de entrada, esta informação é obtida através da ferramenta, no menu "*Wrapper*".

- **Serials** - Contém a informação correspondente a todas as ligações *serial* em que cada *serial* vai ter a seguinte informação, um módulo de saída, um módulo de entrada, um evento de saída, um evento de entrada, se o tipo de ligação entre os dois módulos é feito através de um canal dedicado ou através de um canal partilhado por mais módulos (*type*) é também atribuído um identificador através do campo *id*. Para cada *id* é atribuído um *Rate* em que é possível escolher o *baudrate* e também o *stopbit*. Toda esta informação encontra-se na ferramenta no sub-menu "Serial", que pertence ao menu "*Point-To-Point*".

- **Buses** - Contém pelo menos um *Networkbus* e cada *Networkbus* que representa uma ligação *Bus* tem um *id* e um *busnetworktype* associado. Cada ligação contém um ou mais módulos e as suas respetivas posições, esta informação é obtida através da ferramenta no submenu "*Bus*", que pertence ao menu "*Networking*".

- **Rings** - Contém pelo menos um *Networking* e cada *Networkring* que representa uma ligação em *Ring* e tem um *id* e um *busnetworktype*. Cada ligação contem um ou mais módulos e as suas respetivas posições, esta informação obtida através da ferramenta no sub-menu "*Ring*", que pertence ao menu "*Networking*".

- **DoubleRings** - Contém pelo menos um *Networkdoublering* e cada *Networkdoublering* representa uma ligação *Double Ring* e tem um *id* e um *busnetworktype* associado. Cada ligação contém um ou mais módulos e as suas respetivas posições, esta informação é obtida através da ferramenta no submenu "*Double Ring*", que pertence ao menu "*Networking*".

- **Matrixs** - Contém pelo menos um *Networkmatrix* e cada *Networkmatrix* representa uma ligação em *Matrix* e tem um *id* e um *busnetworktype* associado. Cada ligação contém um ou mais módulos e as suas respetivas posições, esta informação obtida através da ferramenta no submenu "*Matrix*", que pertence ao menu "*Networking*''.

- **Toroidals** - Contém pelo menos um *Networktoroidal* e cada *Networktoroidal* representa uma ligação Toroidal e tem um *id* e um *busnetworktype* associado. Cada ligação

contém um ou mais módulos e as suas respetivas posições, esta informação obtida através da ferramenta no submenu "Toroidal", que pertence ao menu "*Networking*".

- **Plataforms** - todas as plataformas utilizadas e a sua respetiva informação encontra-se dentro deste elemento, informação essa que é composta por um identificador da plataforma (*id*), pelo tipo da plataforma (*typePlatform*), pelo tipo de implementação (*typeImplementation*), pelo domínio temporal (*timedomain*) e pelo nome do módulo que se vai colocar nessa plataforma (módulo).

5

Exemplo de Aplicação

Este capítulo tem como objetivo apresentar o protótipo desenvolvido, através de um exemplo prático, de forma a mostrar toda a sua funcionalidade.

5.1 Exemplo - Nove Carros Sincronizados

Nesta secção, apresenta-se um sistema composto por 9 carros que se movimentam entre os pontos Ai e Bi, sincronizando-se o seu movimento através dos botões de *Go* e *Back*. O sistema é apresentado na figura 5.1

O funcionamento do sistema é bem simples, estando os carros no ponto de partida indicado pelo ponto A e ao se pressionar o botão de *GO*, os carros partirão com destino ao ponto B, ponto esse que indica fim de percurso. Quando os carros se encontram no ponto B, ao se pressionar o botão de *Back*, os carros partirão em direção ao ponto A. Este funcionamento pode ser executado tantas vezes quanto as necessárias.

Figura 5.1: Exemplo - 9 Carros Sincronizados

5.2 Modelação da RdP-IOPT

Através da aplicação IOPT-Tools, o controlador que se pretende obter é modelado em RdP-IOPT, já explicadas anteriormente. A figura 5.2 mostra a modulação efetuada. Na figura não são contemplados os sinais e eventos de saída associados a atuação dos motores, apesar de estes estarem definidos, porque sobrecarregava ainda mais a figura.

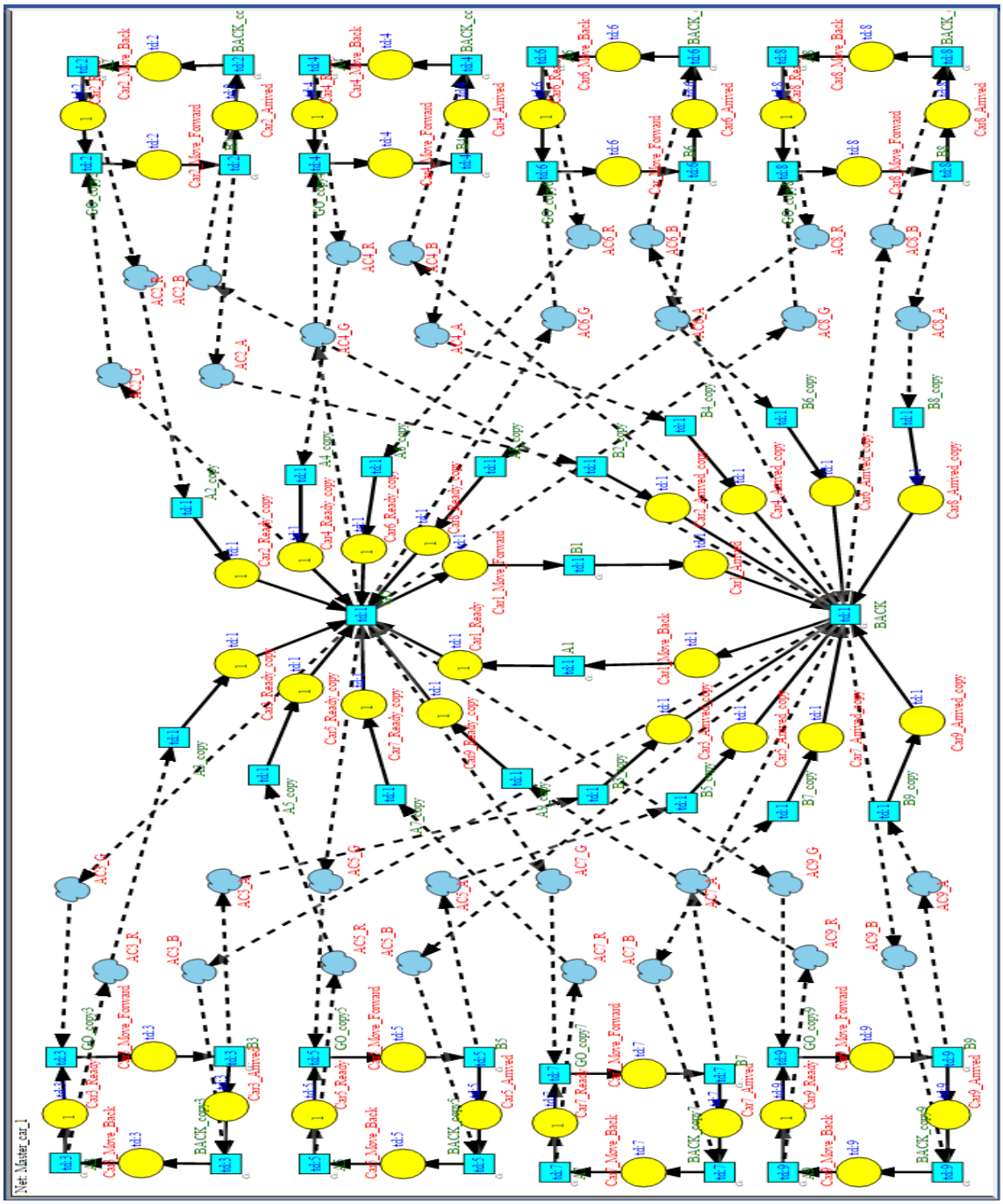


Figura 5.2: RdP IOPT dos nove carros

O controlador do sistema vai ter os seguintes sinais de entrada e saída:

- **Sinais de Entrada**
  - A1, A2, A3, A4, A5, A6, A7, A8 e A9.
  - B1, B2, B3, B4, B5, B6, B7, B8 e B9.
  - Go.

  - Back.

- **Sinais de Saída**
  - M1, M2, M3, M4, M5, M6, M7, M8, M9
  - Dir1, Dir2, Dir3, Dir4, Dir5, Dir6, Dir7, Dir8 e Dir9.

Tendo em conta, a sequência de procedimentos explicada no início deste capitulo, é necessário obter um controlador para cada carro, isto significa que é necessário decompor o modelo apresentado na figura 5.2, em nove submodelos, um para cada carro, para isso foi utiliza a funcionalidade *Decompose GALS*, funcionalidade essa, que faz parte da ferramenta IOPT-Tool. O resultado da utilização da *Decompose GALS* pode ser verificado na figura 5.3.

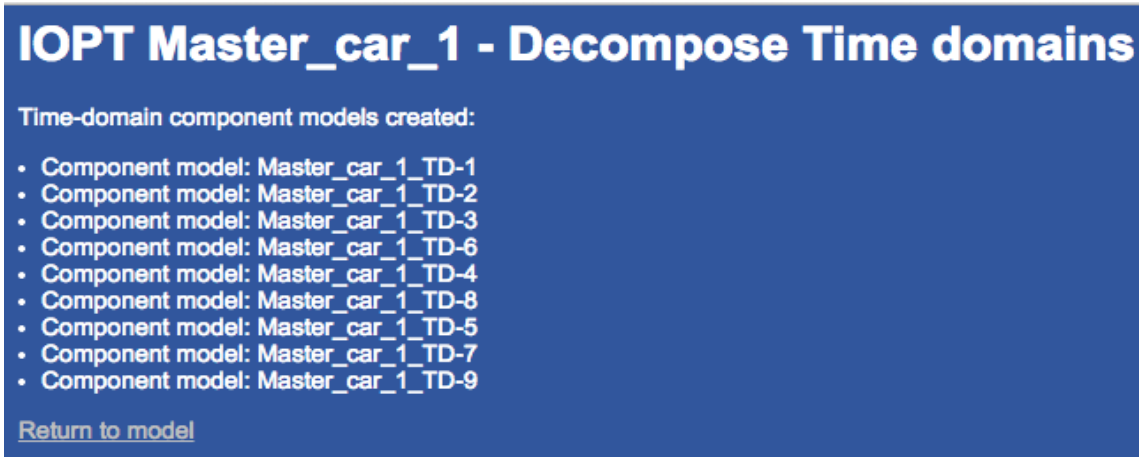


Figura 5.3: Ficheiros gerados após a utlização da funcionalidade *Decomposed GALS*

A decomposição é feita tendo em consideração os diferentes *Time Domains*, como existem nove *Time Domains* diferentes, um por cada carro, logo serão gerados nove submodelos como mostra a figura 5.3. Para não sobrecarregar este capítulo com as figuras dos nove sub-modelos serão mostradas apenas, a figura referente ao carro "1" e ao carro "2".

Na figura 5.4 e na figura 5.5 encontram-se representados os submodelos referentes ao carro "1" e ao carro "2", respetivamente. Em cada uma das figuras encontra-se assinalado com um quadrado a vermelho a identificação dos respectivos *Time Domains*.

Resultante da partição do modelo serão gerados eventos que são associados aos canais de comunicação. Na interligação entre submodelos considera-se a interligação dos eventos de saída e entrada com o mesmo nome, a distinção é feita tendo em conta a direção das setas dos eventos, na figura 5.6 que é um excerto da figura 5.5 é possível verificar isso com detalhe. O *event27* é considerado um evento de entrada pois a seta azul associado ao evento indica que vai entrar na transação, com o *event28* é precisamente o contrario a seta azul encontra-se a sair da transação logo é considerado um eventos de saída. A

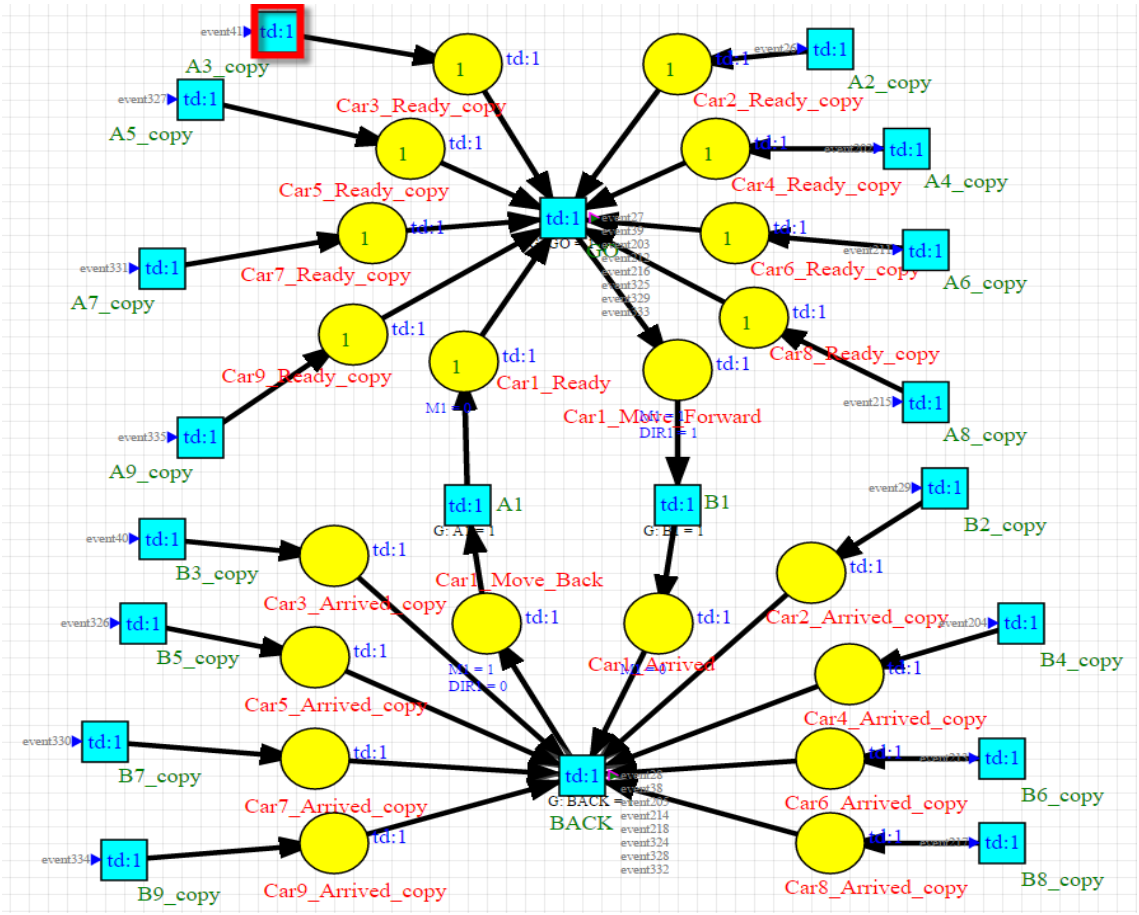


Figura 5.4: RdP IOPT do Carro "1"

funcionalidade *Decompose GALS*, que vai executar a decomposição do modelo, gera estes eventos de comunicação de forma a permitir a interligação entre os diversos submodelos.

Como se pode verificar na figura 5.4, os sinais de entrada *GO* e *Back* estão associados ao carro "1", pois este é considerado o *master*. Ou seja sempre que ocorrer um evento associado ao *GO*, no módulo do carro "1" são gerados eventos de saída para cada um dos outros oito carros, o processo é idêntico para o sinal de *Back*. O eventos de saída associados ao sinal *GO* e ao sinal *Back* são apresentados na tabela 5.1 e na tabela 5.2, respetivamente.

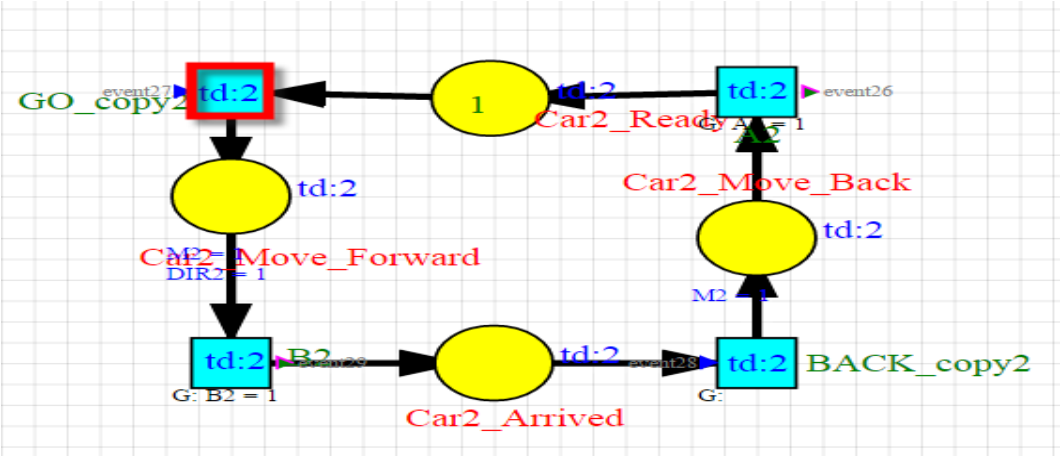


Figura 5.5: RdP IOPT do Carro "2"

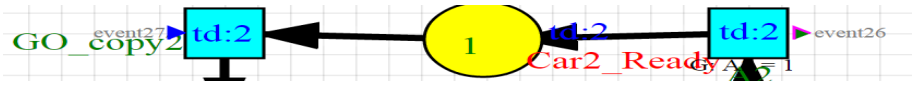


Figura 5.6: Identificação de evento de entrada e saida

	Sinal	Evento de Saída	Evento de Entrada	
Carro1	GO	event27	event27	Carro2
		event39	event39	Carro3
		event203	event203	Carro4
		event325	event325	Carro5
		event212	event212	Carro6
		event329	event329	Carro7
		event216	event216	Carro8
		event333	event333	Carro9

Tabela 5.1: Eventos de saída do carro "1" e os correspondentes eventos de entrada nos outros carros para o sinal *GO*

	Sinal	Evento de Saída	Evento de Entrada	
Carro1	Back	event28	event28	Carro2
		event38	event38	Carro3
		event205	event205	Carro4
		event324	event324	Carro5
		event214	event214	Carro6
		event328	event328	Carro7
		event218	event218	Carro8
		event332	event332	Carro9

Tabela 5.2: Eventos de saída do carro "1" e os correspondentes eventos de entrada nos outros carros para o sinal *Back*

Se o módulo do carro "1" "informou" os restantes módulos da ocorrência de *GO* e *Back*,

com o mesmo principio, os outros módulos também vão ter que avisar o carro "1" quando ocorrerem A2, A3, A4, A5, A6, A7, A8, A9, B2, B3, B4, B5, B6, B7, B8 e B9. A tabela 5.3 mostra os eventos a cada um desses sinais. Os lugares *Car#_Mode_Forward* e *Car#_Move_Back* têm os sinais *M#* e *Dir#* de saída, nos respectivos módulos, associados para indicar movimento.

	Sinal	Evento de Saída	Evento de Entrada	
Carro2	A2	event26	event26	Carro1
	B2	event29	event29	
Carro3	A3	event41	event41	
	B3	event40	event40	
Carro4	A4	event202	event202	
	B4	event204	event204	
Carro5	A5	event327	event327	
	B5	event326	event326	
Carro6	A6	event211	event211	
	B6	event213	event213	
Carro7	A7	event331	event331	
	B7	event330	event330	
Carro8	A8	event215	event215	
	B8	event217	event217	
Carro9	A9	event335	event335	
	B9	event334	event334	

Tabela 5.3: Correspondencia entre os eventos de saida de cada carro com os eventos de entrada do Carro1

5.3 Geração do ficheiro para interligação entre componentes

Nesta seção, será explicado como foi gerado o ficheiro a ser utilizado na ferramenta desenvolvida e também qual a informação que esse ficheiro vai conter.

Como já foi descrito anteriormente no capítulo quatro, na seção 4.6, este ficheiro é criado com informação proveniente no ficheiro PNML do modelo e dos ficheiros PNML dos submodelos. Na figura 5.7 está representada uma parte da informação que o ficheiro gerado contém. Foi apenas apresentada uma parte do ficheiro, pois o ficheiro é algo extenso e para não sobrecarregar este capítulo com imagens muito grandes, achou-se por bem apresentar apenas um excerto do ficheiro.

O intuito da criação deste ficheiro é apenas o de obter um ficheiro simplificado, onde esteja incluída toda a informação necessária, para que o ficheiro quando utilizado pela ferramenta, essa informação seja obtida de uma maneira mais simples. Como não existe nenhuma ferramenta que faça a geração automática deste ficheiro, logo teve de ser criado manualmente.

```
<?xml version="1.0" encoding="UTF-8"?>
<interconnection>
  <distributedmodel filename="Master_car_1.pnml" netname="Master_car_1"/>
  <submodels>
    <submodel submodelid="1" filename="Master_car_1_TD-1.pnml" netname="Master_car_1_TD-1"/>
    <submodel submodelid="2" filename="Master_car_1_TD-2.pnml" netname="Master_car_1_TD-2"/>
    <submodel submodelid="3" filename="Master_car_1_TD-3.pnml" netname="Master_car_1_TD-3"/>
    <submodel submodelid="4" filename="Master_car_1_TD-4.pnml" netname="Master_car_1_TD-4"/>
    <submodel submodelid="5" filename="Master_car_1_TD-5.pnml" netname="Master_car_1_TD-5"/>
    <submodel submodelid="6" filename="Master_car_1_TD-6.pnml" netname="Master_car_1_TD-6"/>
    <submodel submodelid="7" filename="Master_car_1_TD-7.pnml" netname="Master_car_1_TD-7"/>
    <submodel submodelid="8" filename="Master_car_1_TD-8.pnml" netname="Master_car_1_TD-8"/>
    <submodel submodelid="9" filename="Master_car_1_TD-9.pnml" netname="Master_car_1_TD-9"/>
  </submodels>
  <asynchronouschannel acid="26" acname="AC2_R" bound="1">
    <outputevent submodelid="2" eventid="event26"></outputevent>
    <inputevent submodelid="1" eventid="event26"></inputevent>
  </asynchronouschannel>
  <asynchronouschannel acid="27" acname="AC2_G" bound="1">
    <outputevent submodelid="1" eventid="event27"></outputevent>
    <inputevent submodelid="2" eventid="event27"></inputevent>
  </asynchronouschannel>
  <asynchronouschannel acid="28" acname="AC2_B" bound="1">
    <outputevent submodelid="1" eventid="event28"></outputevent>
    <inputevent submodelid="2" eventid="event28"></inputevent>
  </asynchronouschannel>
  <asynchronouschannel acid="29" acname="AC2_A" bound="1">
    <outputevent submodelid="2" eventid="event29"></outputevent>
    <inputevent submodelid="1" eventid="event29"></inputevent>
  </asynchronouschannel>
```

Figura 5.7: Ficheiro INT criado a partir do modelo Principal e dos sub-modelos

Na figura 5.7, é possível verificar que dentro do elemento *submodels* existem nove elementos *sumodel*, em que cada um desses *submodel* corresponde a um Carro, caso existam mais Carros mais elementos *submodel* irão haver, tantos quantos o número de Carros. Cada elemento *asynchronouschannel* corresponde a um canal assíncrono no modelo RdP IOPT, que é representado por uma nuvem de cor azul, isso é possível verificar na figura 5.2. Vão existir tantos elementos *asynchronouschannel* como o número de canais assíncronos existentes no modelo RdP IOPT.

Ainda dentro dos elementos *asynchronouschannel* existem os elementos *outputevent* e *intputevent*, em que cada um corresponde aos eventos de comunicação que são gerados após a partição do modelo, nos nove submodelos, como já foi explicado anteriormente neste capítulo. Assim para cada canal assíncrono do modelo, representado no ficheiro pelo elemento *asynchronouschannel*, ao se fazer a partição do modelo utilizando a funcionalidade *Decompose GALS*, são gerados os submodelos e respetivos eventos de entrada e saída de cada canal assíncrono, esses eventos são representados pelos elementos de *intputevent* e de *outputevent*, respetivamente. Na figura 5.7 encontram-se representados apenas os elementos *asynchronouschannel* referentes à comunicação entre o carro "1" e o carro "2", foi

explicado anterior neste capítulo o porquê da representação de apenas estes dois Carros.

5.4 Implementação

Para a interligação entres os nove submodelos gerados, foram utilizadas as seguintes interligações: *Ring*, *Bus*, e *Serial*, que foram explicadas anteriormente no Capítulo 3. Na figura 5.8 encontram-se representadas as interligações entre os submodelos. Cada *Communication Node i* corresponde a um submodelo, para simplificação considerou-se que *Communication Node 1*, corresponde ao submodelo do carro "1", o *Communication Node 2* ao submodelo do carro "2" e assim sucessivamente.

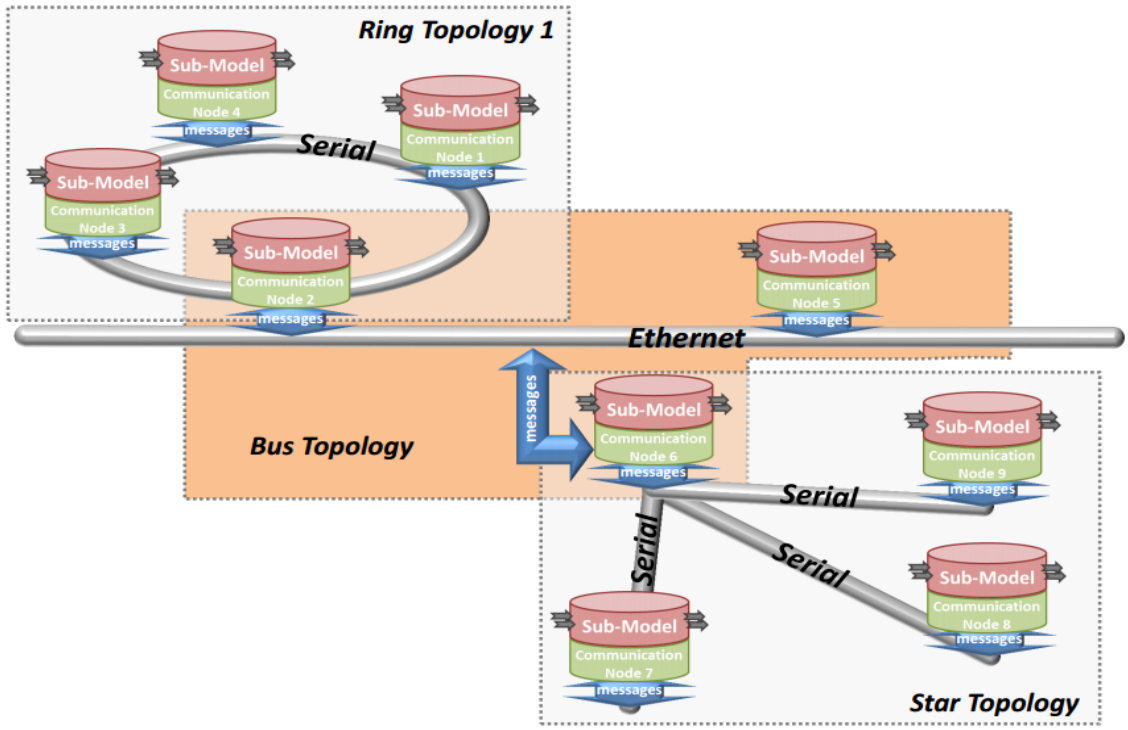


Figura 5.8: Estrutura da ligação entre os diferentes Nós

Visto que já foi gerado o ficheiro que vai ser lido pela aplicação desenvolvida, geração essa que foi explicada na seção 5.3. Pode-se assim passar a sua utilização da ferramenta propriamente dita. Após a escolha do ficheiro, é possível observar todos os eventos de entrada e saída e respetivos módulos de entrada e saída. Esses eventos estão associados aos sinais de *GO*, *Back*, *Ai* e *Bi*, eventos esses que estão representados nas tabelas 5.1, 5.2 e 5.3. É possível observar também outros campos como por exemplo *Nr of Hops*, *Type of Connections* e *Id Network-on-Chip*. Para cada tipo de evento dependo do seu módulo de saida e entrada será necessário definir *Nr Of Hops*, o *Type Of Connections* e dependendo do *Type Of Connections* escolhido (*Bus*, *Ring*, *Double Ring*, *Matrix* e Toroidal) o *Id Network-on-Chip*.

Na figura 5.9 verifica-se que para o *event205* foi escolhido um *Type Of Connections* do tipo *Ring*, o *Id Network-on-Chip* considerado foi um e o *Nr Of Hops* é dois.

Figura 5.9: Ligação dos eventos *event205* e *event324*

O facto do *event205* ter como módulo de saída o carro "1" (*Master_car_1_TD1*) e módulo de entrada o carro "4" (*Master_car_4_TD4*), e como sabemos que o carro "1" e o carro "4" se encontram interligado em *Ring*, isso é verificado na figura 5.8, logo o *Type of Connections* a ser utilizado será o *Ring*. O *Nr Of Hops* considerado é dois, pois o módulo de entrada e de saída do evento encontram-se no mesmo *Ring*. Isso é verificado pelo facto de o *Id Network-on-Chip* ter sempre o mesmo valor, nesta caso "1", a quando da utilização *Type of Connections* do tipo *Ring*. Caso houvesse *Type of Connections* do tipo *Ring* mas com um *Id Network-on-Chip* com um valor diferente significava que existiriam pelo menos dois *Rings*.

O *event324* apresentado na figura 5.9 que tem como módulos de saída o carro "1" (*Master_car_1_TD1*) e módulo de entrada o carro "5" (*Master_car_4_TD5*), verificamos através da figura 5.8 que estes dois módulos não se encontram ligados diretamente, é necessário achar *nodes* intermédios para os poder interligar. Neste caso em especifico o *node1* vai enviar para o *node2* através de *Ring* e o *node2* vai enviar para o *node5* através de *Bus*. O *Nr Of Hops* é considerado três porque são considerados três *nodes*, sendo eles o *node1*, *node2* e o *node5*.

Para o caso apresentado na figura 5.10 verifica-se *Nr Of Hops* igual a quatro isto deve-se ao facto de modulo de saída o carro "1" (*Master_car_1_TD1*) e modulo de entrada o carro "7" (*Master_car_7_TD7*) não se interligarem diretamente, verificação feita através da figura 5.8. Para que a comunicação seja possível, foram definidos os *nodes* intermédios que permitem essa interligação, para este caso, o *node1* vai enviar para o *node2* através de *Ring* e o *node2* vai enviar para o *node6* através de *Bus* e o *node6* vai enviar para o *node7* através de *Serial*.

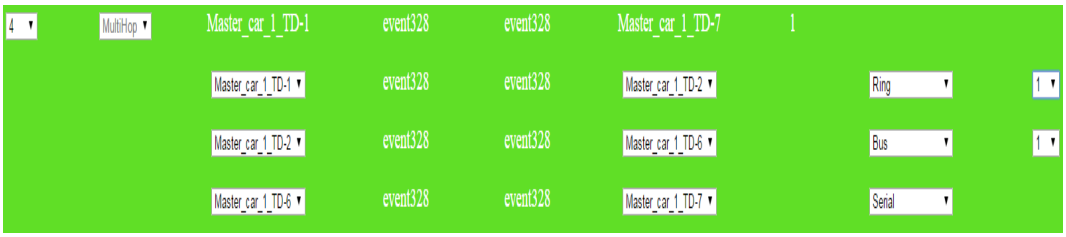


Figura 5.10: Ligação do evento *event328*

Seguidamente, atribui- se a cada módulo, a plataforma e o *Id Platform* e também o respetivo *Time Domain*. O *Type Implementation* vai depender da plataforma utilizada. A figura 5.11 apresenta para cada módulo, a plataforma, o *Id Platform*, o *Type Implementation* e o *TimeDomain*, num total de nove módulos, em que cada um corresponde a um carro.

Id Platform	Type Platform	Type Implementation	Time Domain	Modulo	Delete
U1	Spartan3 Start Kit	VHDL	t_1	Master_car_1_TD-1	☐
U1	Spartan3 Start Kit	VHDL	t_2	Master_car_1_TD-2	☐
U1	Spartan3 Start Kit	VHDL	t_3	Master_car_1_TD-3	☐
U1	Spartan3 Start Kit	VHDL	t_4	Master_car_1_TD-4	☐
U2	Virtex7	VHDL	t_5	Master_car_1_TD-5	☐
U3	ZedBoard (ZYNQ)	VHDL	t_6	Master_car_1_TD-6	☐
U3	ZedBoard (ZYNQ)	VHDL	t_7	Master_car_1_TD-7	☐
U3	ZedBoard (ZYNQ)	VHDL	t_8	Master_car_1_TD-8	☐
U3	ZedBoard (ZYNQ)	VHDL	t_9	Master_car_1_TD-9	☐

Figura 5.11: Plataformas utilizadas no exemplo de aplicação

Na figura 5.12, é ilustrado todos os eventos e respetivos módulos de entrada e saída que foram interligados através de *Serial*. Para cada um desses eventos é possível escolher se esse evento é dedicado (*Dedicated*) ou partilhado (*Shared*), caso seja um evento dedicado significa que o canal de comunicação só permite a a passagem desse mesmo evento entre o módulo de saída e de entrada, para o caso de ser partilhado significa que no canal de comunicação pode passar mais que um evento entre o módulo de saída e o módulo de entrada. Para cada canal de comunicação dedicado ou partilhado é atribuído um identificar numérico (*Id Serial*) que permite identificar com maior facilidade esses canais de comunicação.

Figura 5.12: Ligações Série efetuadas no exemplo de aplicação

Para cada um desses canais de comunicação é possível escolher o *Baud Rate* e o *Stop Bit*. O valores de *Baud Rate* possíveis de escolher são 2400, 9600 e 115200 enquanto para o *Stop Bit* são 1, 1.5 e 2. *Baud Rate* refere-se ao número de vezes que um sinal ou um símbolo muda a sua ocorrência por segundo, sendo que um símbolo é uma de muitas mudanças ao nível da frequência, voltagem e fase [29]. Em transmissões assíncronas o *Stop Bit* representa o numero de bits (1, 1.5 ou 2) que são acrescentados ao final da trama para indicar o final desta, [28].

Para o *Bus* é necessário definir as posições dos módulos na respetiva interligação. Para o caso do *Bus* que é ilustrado na figura 5.13 definiu-se que o modulo *Master_car_2_TD2* (Carro2) se encontra na posição um, o modulo *Master_car_5_TD5* (Carro5) na posição dois e *Master_car_6_TD6* (Carro6) na posição três.

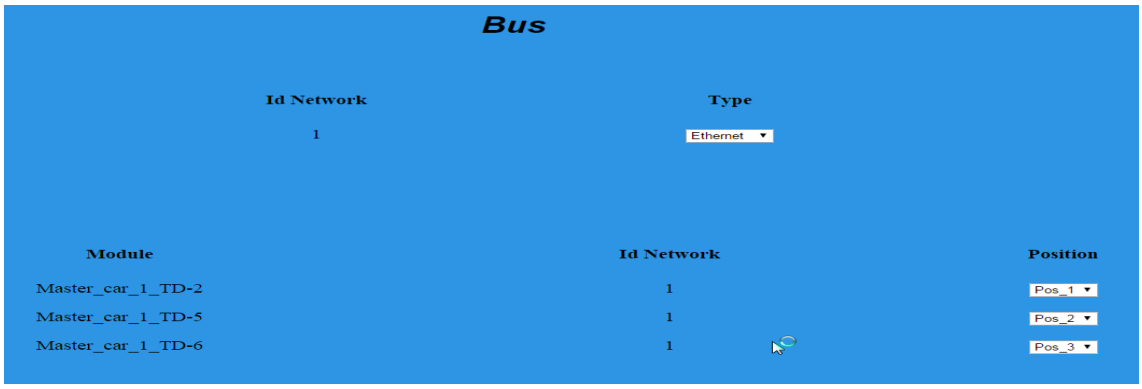


Figura 5.13: Ligação *Bus* efetuada no exemplo de aplicação

Cada *Id Network* permite identificar cada um dos diferentes *buses* sendo possível escolher qual o tipo de ligação para cada um desses *buses*, os tipos de ligação são os seguintes: internet, *Wifi*, *Bluetooth*, ZigBee, I2C e SPI. No caso deste exemplo de aplicação apenas se utilizou um *Bus* sendo o *Id Network* de um e o tipo de ligação a Internet.

Para o caso do *Ring* que é ilustrado na figura 5.14 definiu-se que o modulo *Master_car_1_TD11* (Carro1) se encontra na posição um, o modulo *Master_car_2_TD2* (Carro2) na posição dois, na posição três o modulo *Master_car_3_TD3* (Carro3) e na posição quatro *Master_car__TD4* (Carro4).

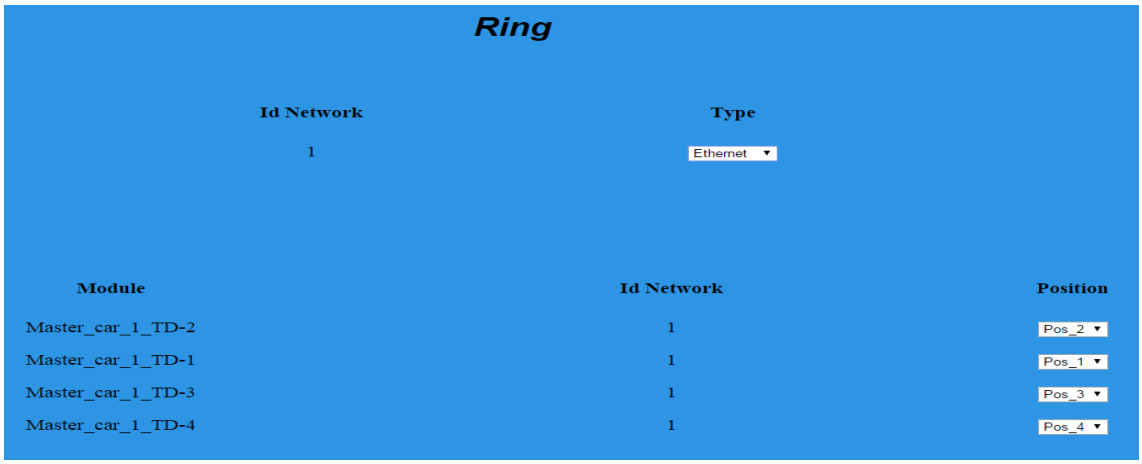


Figura 5.14: Ligação em Ring efetuada no exemplo de aplicação

Cada *Ring* tal como o *Bus* tem um *Id Network* que permite identificar cada um dos diferentes *Rings*, sendo possível também escolher qual o tipo de ligação para cada deles. Os tipos de ligação são os seguintes: *Serial* e *Synchronous Serial*. No caso deste exemplo de aplicação apenas se utilizou um *Ring* sendo o *Id Network* de um e o tipo de ligação a

Internet.

Depois de definidos todos os passos anteriormente explicados nesta secção o utilizador poderá gerar o ficheiro com toda essa informação. A figura 5.15 mostra na aplicação onde esse ficheiro pode ser gerado.

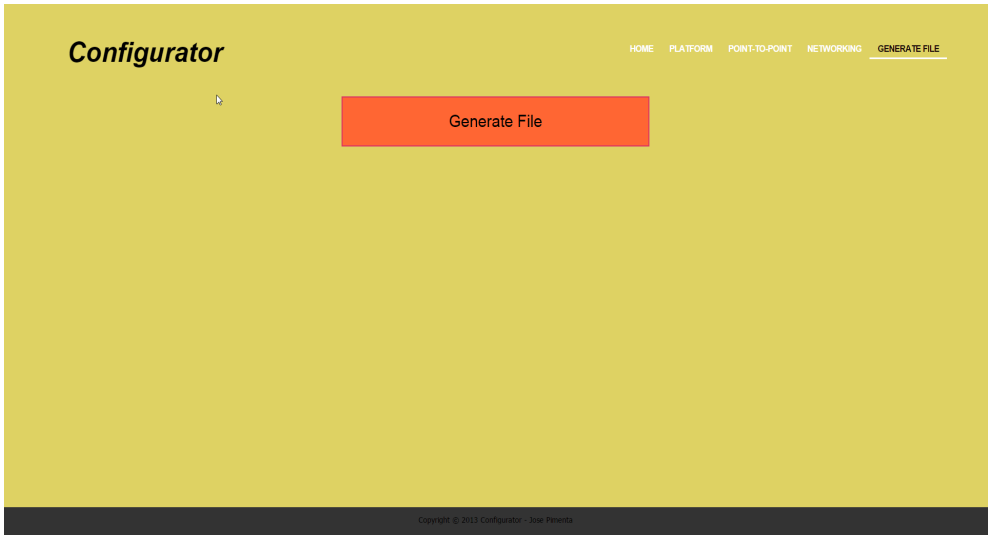


Figura 5.15: Menu *Generate File*

Por último é gerado o ficheiro que tem o formato apresentado na figura 5.16. Apenas é apresentado um excerto pois o ficheiro gerado era algo grande sendo apresentado na sua totalidade no apêndice B.

```
    <Serial outputmodulo="Master_car_1_TD-6" outputevent="event333" inputevent="event333" inputmodulo="Master_car_1_TD-9" type="Shared" id="8"/>
    <Serial outputmodulo="Master_car_1_TD-9" outputevent="event334" inputevent="event334" inputmodulo="Master_car_1_TD-6" type="Shared" id="8"/>
    <Serial outputmodulo="Master_car_1_TD-9" outputevent="event335" inputevent="event335" inputmodulo="Master_car_1_TD-6" type="Shared" id="8"/>
    <Rate serialid="0" baudrate="2400" stopbit="1"/>
    <Rate serialid="4" baudrate="2400" stopbit="1"/>
    <Rate serialid="8" baudrate="2400" stopbit="1"/>
  </Serials>
  <Buses>
    <Networkbus id="1" busnetworktype="Ethernet">
      <Bus modulo="Master_car_1_TD-2" position="Pos_1"/>
      <Bus modulo="Master_car_1_TD-5" position="Pos_2"/>
      <Bus modulo="Master_car_1_TD-6" position="Pos_3"/>
    </Networkbus>
  </Buses>
  <Rings>
    <Networkring id="1" ringnetworktype="Serial">
      <Ring modulo="Master_car_1_TD-2" position="Pos_2"/>
      <Ring modulo="Master_car_1_TD-1" position="Pos_1"/>
      <Ring modulo="Master_car_1_TD-3" position="Pos_3"/>
      <Ring modulo="Master_car_1_TD-4" position="Pos_4"/>
    </Networkring>
```

Figura 5.16: Excerto do ficheiro gerado pela ferramenta desenvolvida

6

Conclusões e perspetivas futuras

Neste trabalho, é apresentada uma ferramenta computacional que permite a geração de um ficheiro de configuração para plataformas específicas através de redes de Petri (RdP) permitindo a integração de componentes que foram previamente definidos. Uma vez que os modelos RdP se encontram descritos no formato PNML (*Petri Net Markup Language*), a ferramenta desenvolvida deverá identificar os eventos de entrada e saída de cada componente, bem como permitir escolher a plataforma em que cada componente deve ficar, permitindo que o utilizador defina a forma de interligação entre os eventos: se utilizando *Bus, Ring, Double Ring, Matrix, Toroidal*, ou *Wrappers* ou através de ligação direta (*Serial*).

Selecionar qual a informação que seria necessária para criar o ficheiro que é lido pela ferramenta desenvolvida foi um dos desafios desta dissertação, pois à medida que se ia desenvolvendo a ferramenta este ficheiro foi sendo alterado devido à informação que ia sendo necessário acrescentar a este ficheiro. A estrutura desse mesmo ficheiro também foi um processo importante, porque ao ser lido convém que tenha uma estrutura relativamente simples e não complexas, pois quanto mais complexa maior será o tempo necessário para a sua leitura.

À medida que se foi desenvolvendo a ferramenta, vários problemas foram surgindo. Inicialmente o facto de não se ter considerado que um evento podia passar por diferentes componentes até chegar ao seu destino fez com que se tivesse que alterar a ferramenta como ela estava inicialmente projetada.

Um dos desafios foi tentar apresentar de uma maneira simples toda a informação que o utilizador necessita de visualizar/preencher. Para isso foi necessário estudar com detalhe os meios de ligação utilizados, sendo eles: *Serial, Wrapper, Bus, Ring, Double Ring, Matrix*

e Toroidal.

Tendo em conta os objetivos propostos inicialmente, as várias necessidades que foram detetadas ao longo do desenvolvimento deste trabalho, pode considerar-se que a ferramenta desenvolvida no âmbito desta dissertação é um contributo para a utilização das RdP-IOPT no desenvolvimento de sistemas assíncronos. O resultado obtido foi bastante satisfatório.

O trabalho desenvolvido contribuiu para duas comunicações a eventos técnico-ciêntificos: a primeira integrada com outros trabalhos desenvolvidos no âmbito das IOPT-Tools e apresentada nas jornadas REC2012 [21] e a segunda também integrada no âmbito das IOPT-Tools na ISIE2013 [22].

Integrar a ferramenta desenvolvida no ambiente IOPT-Tools será também um objetivo de futuro, pois esta ferramenta será mais uma para completar todas as outras que já existem no ambiente IOPT-Tools.

O passo seguinte é o desenvolver nesta mesma ferramenta, uma parte em que seja possível ao utilizador atribuir os pins em cada uma das plataformas utilizadas, de maneira automática ou manualmente.

Através do ficheiro gerado pela ferramenta, o objetivo seguinte será a implementação de uma nova ferramenta que permita a geração automática de código que permita implementar os diferentes componentes nas respetivas plataformas.

A criação de uma ferramenta que gere o ficheiro que inicialmente vai ser lido pela ferramenta desenvolvida será também necessário, pois esse ficheiro foi inicialmente gerado à mão.

Esta própria ferramenta pode ser melhorada permitindo a ligação entre componentes em vez de se considerar apenas a ligação entre eventos. Esta funcionalidade seria útil caso o utilizador quisesse que todos os eventos entre dois componentes fossem ligados da mesma forma. Assim pouparia tempo em vez de os ligar evento a evento.

Bibliografia

[1] *Arduino Due*. Accessed: 20 Agosto 2015. URL: https://www.arduino.cc/en/Main/ArduinoBoardDue.

[2] *Arduino Duemilanove*. Accessed: 20 Agosto 2015. URL: https://www.arduino.cc/en/Main/ArduinoBoardDuemilanove.

[3] *Arduino Uno*. Accessed: 20 Agosto 2015. URL: https://www.arduino.cc/en/Main/arduinoBoardUno.

[4] J. Billington, S. Cristensen, K. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno e M.Weber. "The Petri Net Markup Language: Concepts, Technology, and Tools". Em: In Proceeding of the 24th International Conference on Application e Theory of Petri Nets, June 2009.

[5] D. M. Chapiro. "Globally-Asynchronous Locally-Synchronous Systems". Tese de doutoramento. Stanford University, 1984.

[6] A. Costa e L. Gomes. "Petri net partitioning using net splitting operation". Em: In 7th IEEE International Conference on Industrial Informatics (INDIN 2009), 2009.

[7] *EIA - 232 Bus*. Accessed: 10 Jul 2013. URL: http://www.interfacebus.com/Design_Connector_RS232.html.

[8] S. Erno, K. Ari e D. H. Timo. "On network-on-chip comparison". Em: Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods e Tools: IEEE Computer Society, 2007.

[9] H. A. Ferreira. "Petri Nets Based Component Within Globally Asynchronous Locally Synchronous systems". Tese de mestrado. Faculdade de Ciência e Tecnologia da Universidade Nova de Lisboa, 2010.

[10] R. Ferreira, A. Costa e L. Gomes. "Intra- and intercircuit network for Petri nets based components". Em: Industrial Electronics (ISIE), International Symposium on, 2011.

[11] R. W. Ferreira. "Comunicação intra- e inter-circuito de componentes especificos com Redes de Petri". Tese de mestrado. Faculdade de Ciência e Tecnologia da Universidade Nova de Lisboa, 2010.

[12] *FORDESIGN – Formal Methods for Embedded Systems Co-Design*. Accessed: 15 Jan 2013. FCT - Fundação para a Ciência e a Tecnologia, 2007. URL: `www.uninova.pt/fordesign`.

[13] L. Gomes, J. Barros, A. Costa e R. Nunes. "The Input-Output Place-Transition Petri Net Class and Associated Tools". Em: In 5th IEEE International Conference on Industrial Informatics INDIN 2007, 2007.

[14] E. R. Harold e W. S. Means. *XML in a nutshell*. Terceira. 2006.

[15] P. Heitlinger. *O Guia Pratico da XML*. Primeira. November 2001.

[16] D. Inc. "Digilent Basys2 Board Reference Manual". Em: Digilent Inc., 2010.

[17] *IOPT Tools User Manual*. Accessed: 15 Fev 2014. URL: `http://gres.uninova.pt/iopt_usermanual.pdf`.

[18] Microchip. "PIC18F2525/2620/4525/4620 Data Sheet". Em: Microchip Technology, Inc, 2004.

[19] F. Moutinho e L. Gomes. "'State Space Generation Algorithm for GALS Systems Modeled by IOPT Petri Nets". Em: Proceeding of the 37th Annual Conference of hte IEEE Industrial Electronics Society, Melbourne, Australia.

[20] F. Moutinho e L. Gomes. "State Space Generation for Petri nets-based GALS Systems". Em: Industrial Technology (ICIT), IEEE Internation Conference on, Atenas, Grécia, 2012.

[21] F. Moutinho, J. Pimenta e L. Gomes. "Dimensionamento da infraestrutura de comunicação em sistemas GALS especificados através de redes de Petri". Em: REC'2012 - VIII Jornadas sobre Sistemas Reconfiguráveis, 2012.

[22] F. Moutinho, J. Pimenta e L. Gomes. "Configuring communication nodes for networked embedded systems specified by Petri nets". Em: ISIE'2013 - 22nd IEEE International Symposium of Industrial Electronics, 2013.

[23] F. Moutinho, J. Pimenta e L. Gomes. "Dimensionamento de buffers para redes ponto a ponto de sistemas GALS especificados através de redes de Petri". Em: REC'2013 - 9th Portuguese Meeting on Reconfigurable Systems, 2013.

[24] T. Murata. "Petri net: Properties, Analysis and Applications". Em: Proceedings of the IEEE (Volume: 77, Issue: 4), Abril 1989.

[25] J. Muttersbach, T. Villiger e W. Fichtner. "Practical design of globally-asynchronous locally-synchronous systems". Em: Advanced Research in Asynchronous Circuits e Systems, (ASYNC 2000) Proceedings. Sixth International Symposium on, 2000, pp. 52–59.

[26] J. Nurmi. “Network-on-Chip: A New Paradigm for System-on-Chip Design”. Em: System-on-Chip, International Symposium on, 2005, pp. 2–6.

[27] J. Spars e S. Furber. *Principles of assynchronous circuits design - A System Perspective*. 2001.

[28] *Start-Stop Making Sense*. Accessed: 20 Jun 2014. URL: `http://www.yale.edu/pclt/COMISDN/STRTSTOP.HTM`.

[29] *What's The Difference Between Bit Rate And Baud Rate?* Accessed: 16 Jun 2014. URL: `http://electronicdesign.com/communications/what-s-difference-between-bit-rate-and-baud-rate`.

[30] Xilinx. “Xilinx Synthesis Technology (XST) User Guide”. Em: Xilinx, Inc, 2002.

[31] Xilinx. “KC705 Evaluation Board for the Kintex-7 FPGA”. Em: Microchip Technology, Inc, 2004.

[32] Xilinx. “VC707 Evaluation Board for the Virtex-7 FPGA”. Em: Microchip Technology, Inc, 2004.

[33] Xilinx. “Spartan-3 Starter Kit Board User Guide”. Em: Xilinx, Inc, 2014.

[34] ZedBoard. “ZedBoard Getting Started Guide”. Em: ZedBoard.

Apêndice A

A.1 Plataformas Suportadas

Na ferramenta desenvolvida, foram consideras as seguintes plataformas de implementação:

- **Spartan-3 Starter Kit Board** - A placa do Kit da Xilinx, representada na figura A.1, retirada de [33], é constituída por uma FPGA Xilinx Spartan-3 (XC3S200), uma plataforma flash XCF02S Xilinx, duas memórias SRAM (256K x 16), quatro botões de pressão, oito interruptores, oito LEDS, quatro *displays* de 7-segmentos, RS232, JTAG, PS/2 (rato ou teclado), porta VGA três portos de expansão que permitem a possibilidade de interligação da FPGA a outros dispositivos.

- **Virtex-7** - A placa VC707 da Xilinx, representada na figura A.2, retirada de [32], é constituída por uma FPGA Virtex-7 (XC7VX485T-2FFG1761C), memória de 1GB DD3, USB 2.0 *transceiver*, ethernet PHY de 3 velocidades 10/100/1000, ligação HDMI, 5 botões direcionais, LCD para display de caracteres (16 caracteres x 2 linhas) entre muitos outros.

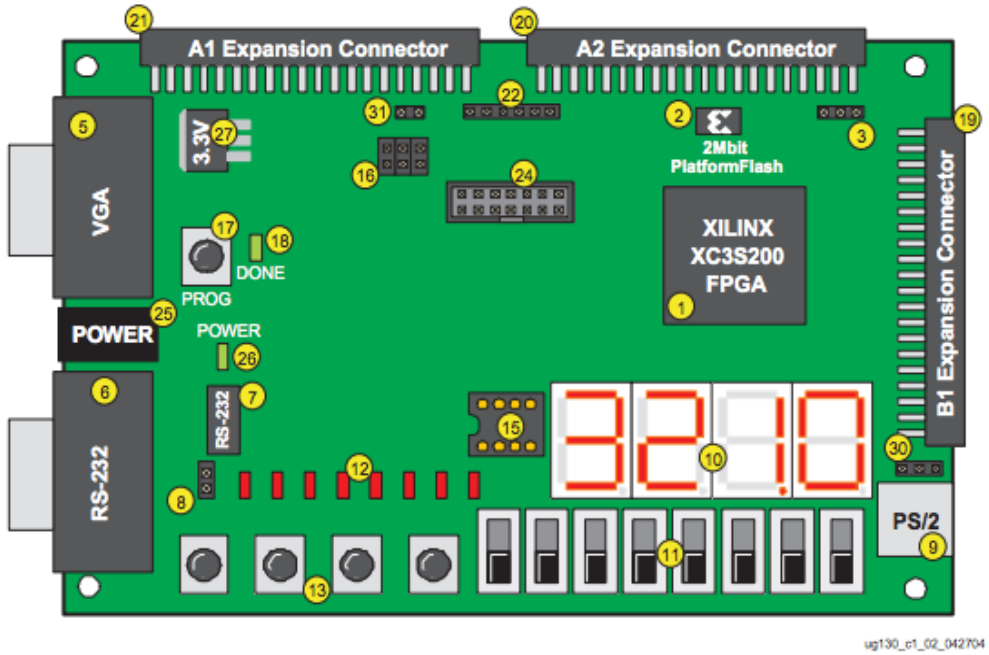


Figura A.1: Xilinx Spartan-3 Starter Kit Board (Vista de Cima) [33]

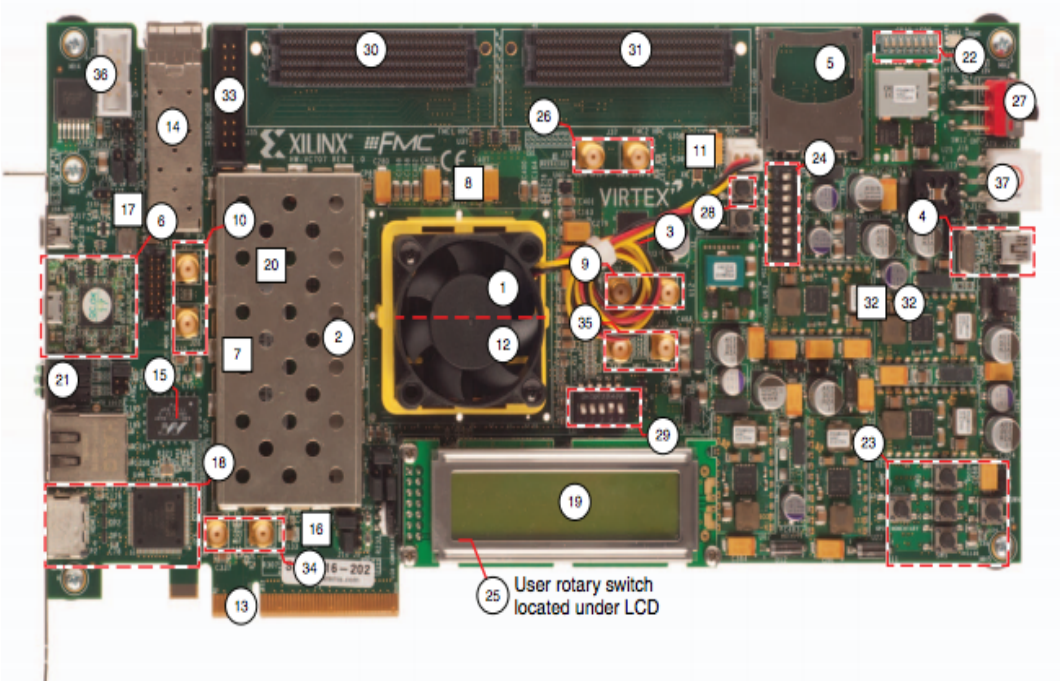


Figura A.2: Placa VC707 [32]

- **ZedBoard** - A placa ZedBoard, representada na figura A.3, retirada de [34], é constituida por um processador Xilinx Zynq-7000, memoria de 512MB DD3, cartão SD (*Secure Digital*) ate 4GB, memoria de 256MB Quad-SPI (*Serial Peripheral Interface*),

ethernet de 3 velocidades 10/100/1000, ligação HDMI, *dysplay* OLED (128X32 pixels), etc.

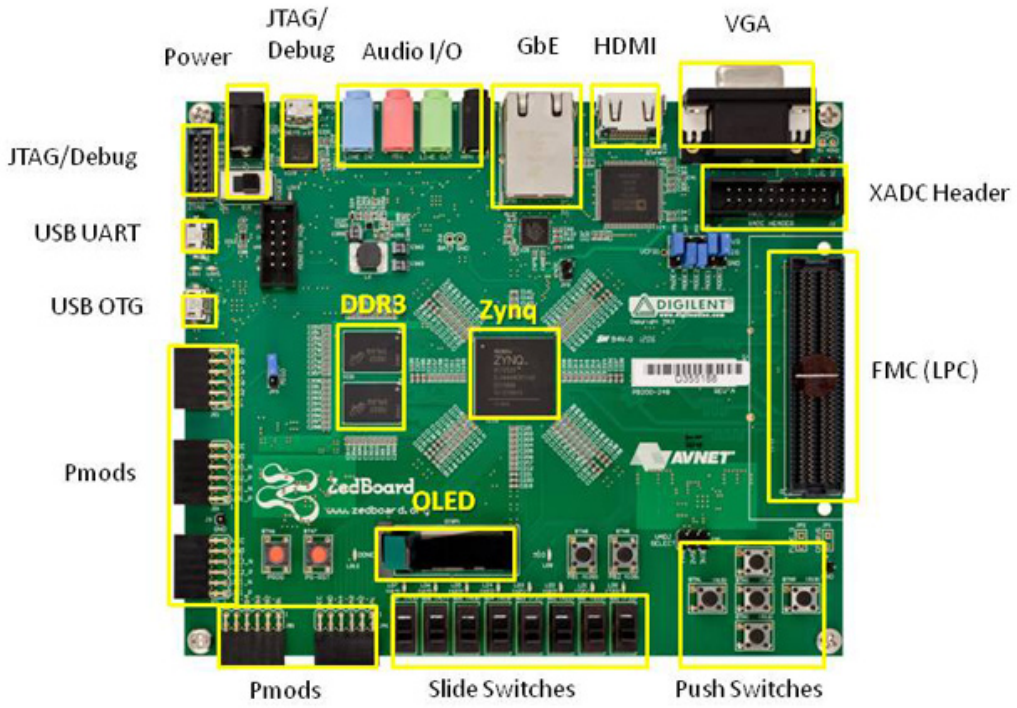


Figura A.3: ZedBoard [34]

- **PIC18F4620** -Devido a ser um microcontrolador de baixo custo, notas de aplicação extensas e ferramentas de desenvolvimento gratuitas tornou-se muito popular.

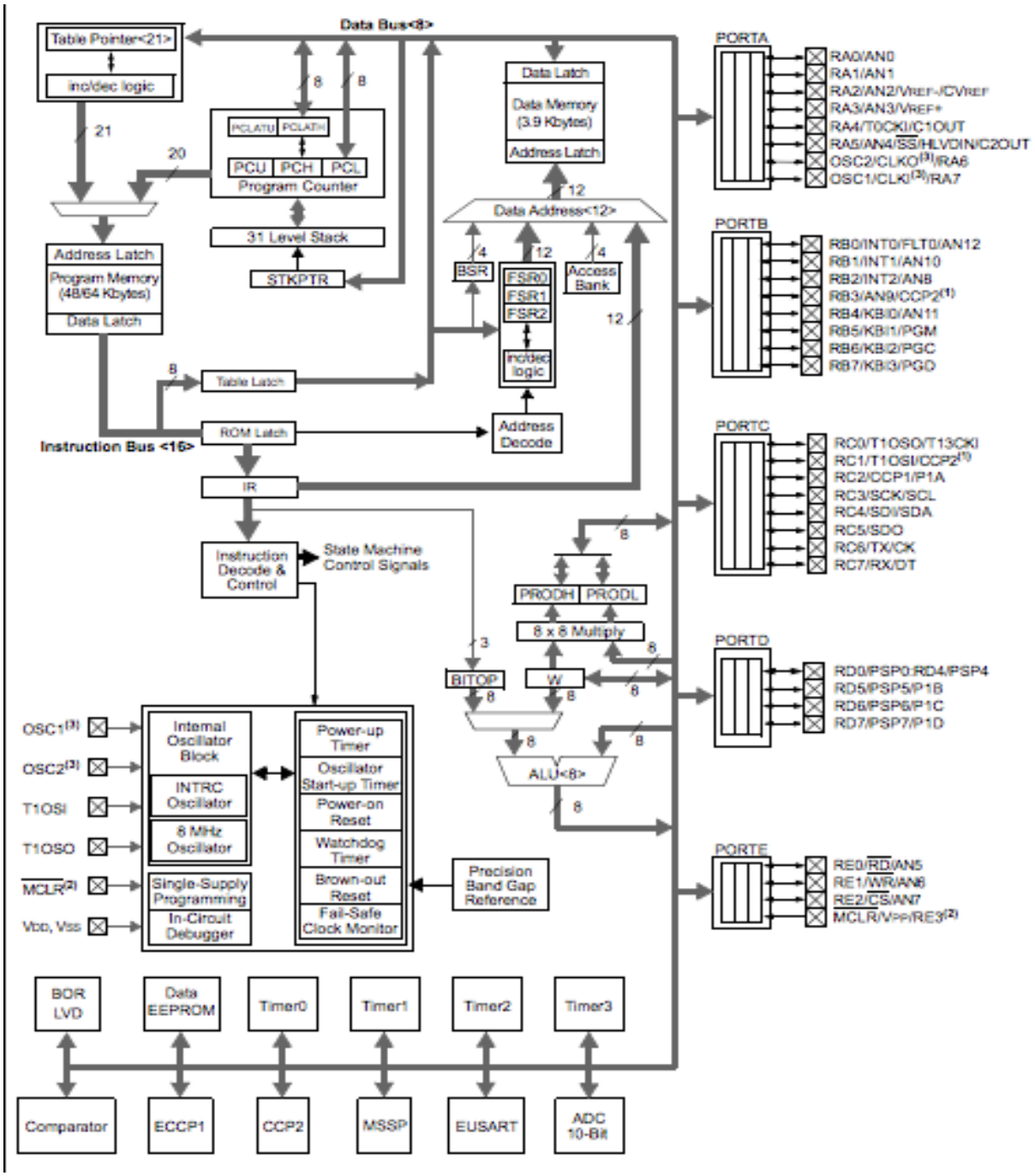


Figura A.4: Diagrama de blocos do Pic18F4620 [18]

O PIC18F4620 possui 64 KB de memória de programa *flash*, 4KB de memória RAM e 1KB de EEPROM, sem possibilidade de expansão. No circuito existem vários periféricos incluídos, como se pode ver na figura A.4 retirada de [18], tais como: temporizadores, comparadores, entradas e saídas para sinais analógicos e digitais, etc.

- **BASYS2** - A placa BASYS2 é uma plataforma de implementação que qualquer pessoa pode utilizar para construir circuitos digitais. A BASYS2 suporta tanto básicos dispositivos lógicos como controlos complexos. Contém também uma largo conjunto de dispositivos de *I/O* e todos os circuitos suportados por uma FPGA estão incluídos nesta placa. Existem quatro conetores de expansão que permitem acrescentar novas placas tais como *bredboards* entre outros dispositivos.

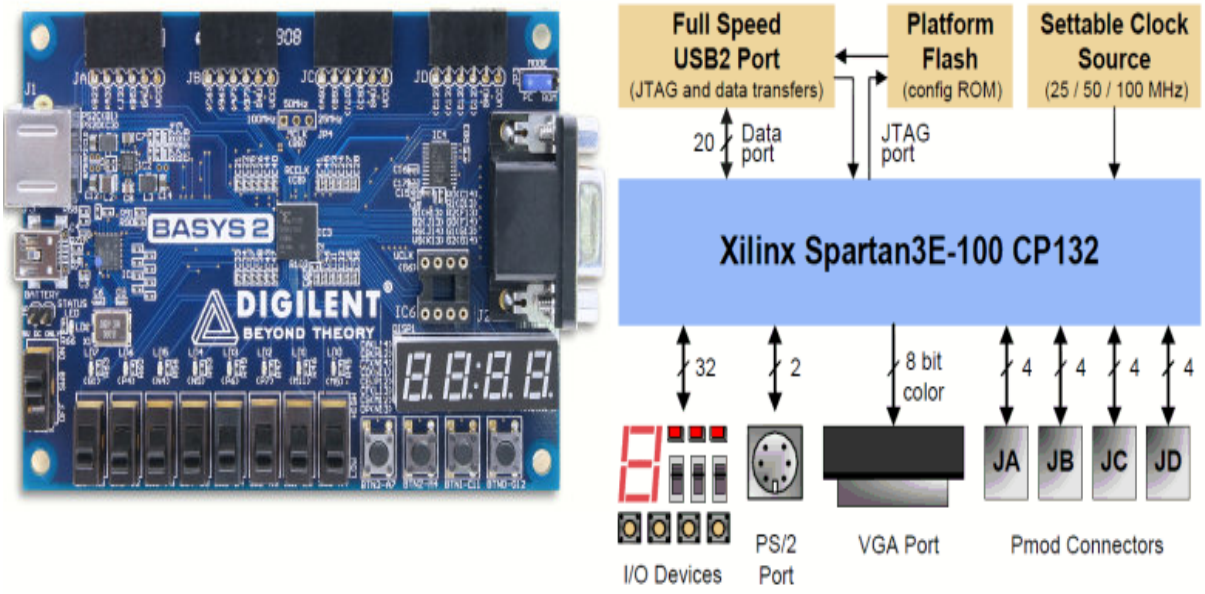


Figura A.5: BASYS2 [16]

- **Kintex7** - A kintex7 FPGA KC705 é uma placa que permite obter alta performance aplicacionais e também alta largura de banda nos circuitos ai implementados. A Kintex7 é constituída por uma memoria DDR3, por portas series normalizadas do tipo PCI *express*, XAUI entre outros, o numero de I/O podem ser aumentados através do uso da FMC.

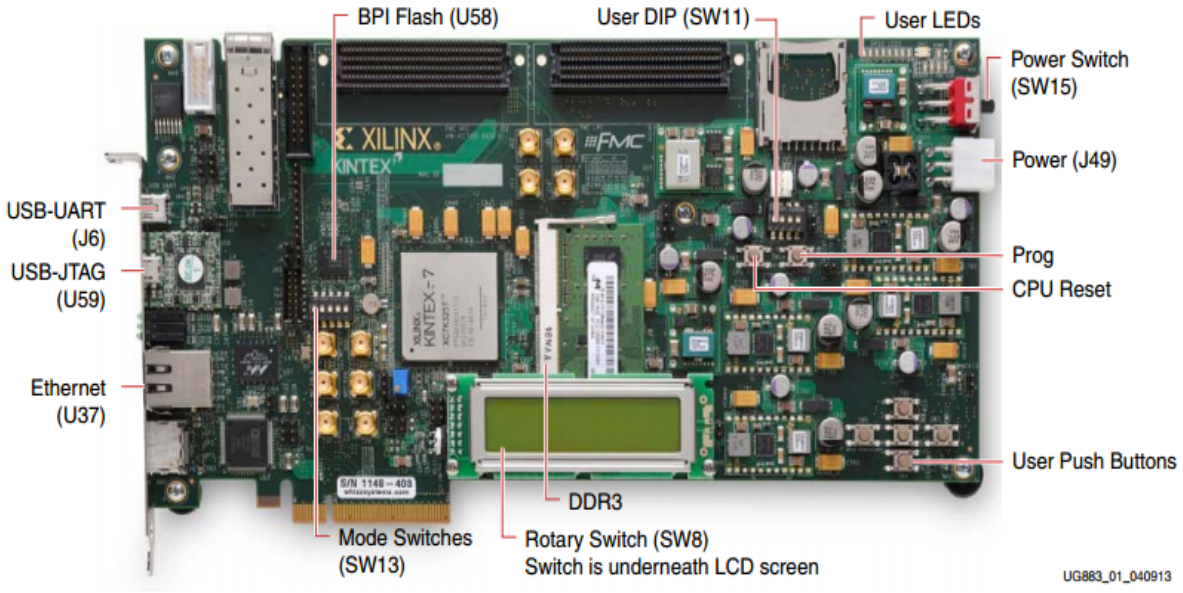


Figura A.6: Kintex7 FPGA KC705 [31]

- **Arduino Deumilanove** - O Arduino Deumilanove é um placa que utiliza um microcontrolador do tipo ATmega168 ou ATmega328. Tem 14 pins digitais de *input/output*, 6 analógicos *inputs*, ligação USB, botão de *reset*, entre outras características. Contém tudo o que é necessário para suporte ao microcontrolador. Basta ligar ligá-lo ao computador através de USB e está preparado para começar.

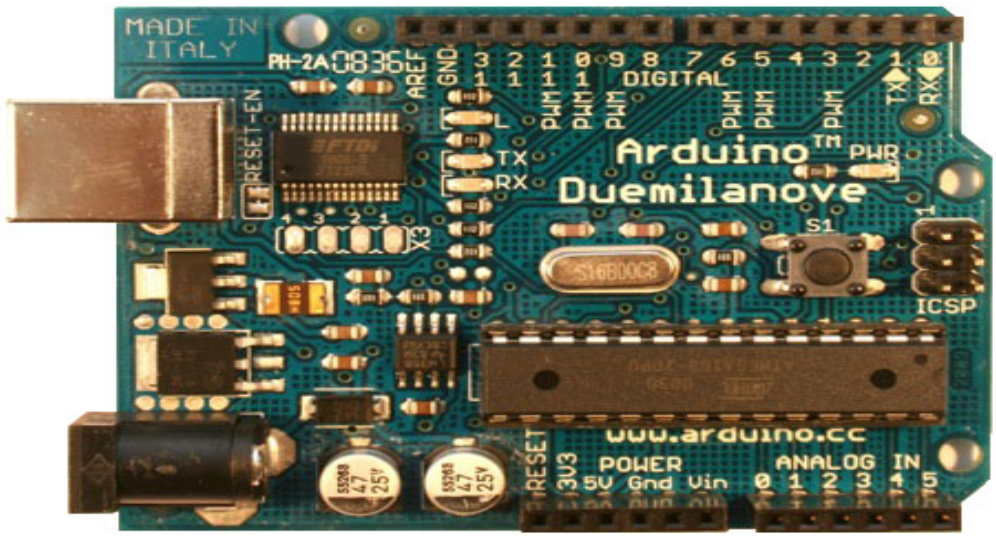


Figura A.7: Arduino Deumilanove [2]

Microcontroller	ATmega168
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	16 KB (ATmega168) or 32 KB (ATmega328) of which 2 KB used by bootloader
SRAM	1 KB (ATmega168) or 2 KB (ATmega328)
EEPROM	512 bytes (ATmega168) or 1 KB (ATmega328)
Clock Speed	16 MHz

Figura A.8: Características do Arduino Deumilanove [2]

- **Arduino Uno** - O Arduino Uno é um placa que utiliza um microcontrolador do tipo ATmega328P. Tem 14 pins digitais de *input/output*, 6 analógicos *inputs*, ligação USB, botão de *reset* entre outras caracteristicas. Contém tudo o que é necessário para suporte ao microcontrolador. Basta ligar ligá-lo ao computador através de USB e está preparado para começar.

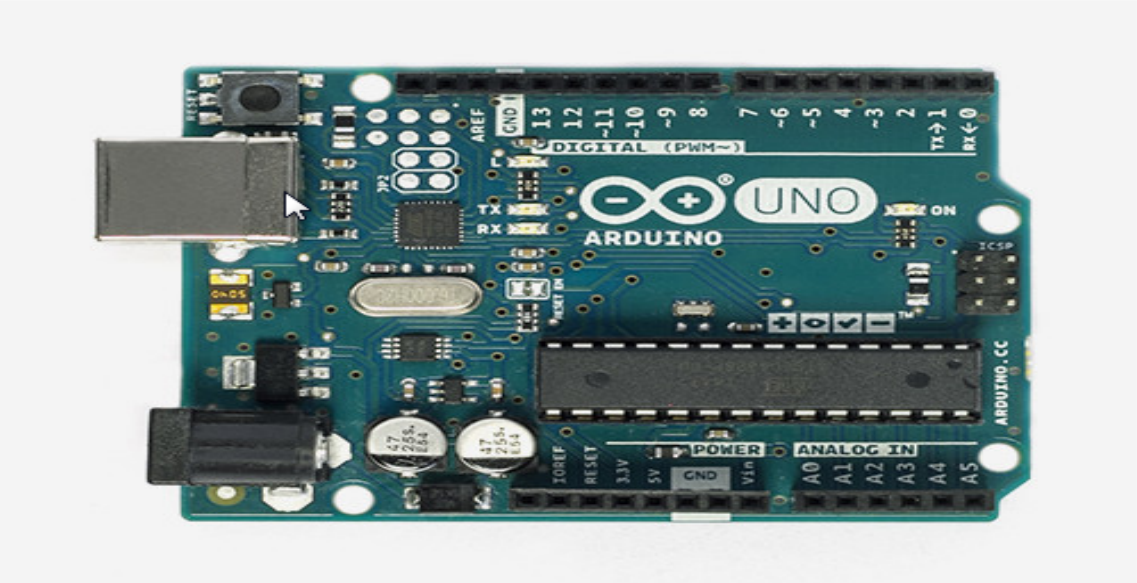


Figura A.9: Arduino Uno [3]

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Figura A.10: Características do Arduino Uno [3]

- **Arduino Due** - O Arduino Due é um placa que utiliza um microcontrolador do tipo Atmel SAM3X8E ARM Cortex-M3 CPU. É o pirmeiro Arduino baseado num microcontrolador de 32 bits ARM core. Tem 54 pins digitais de *input/output*, 12 analógicos *inputs*, 4 UARTS (*hardware serial ports*), ligação USB, botão de *reset* entre outras caracteristicas. Contém tudo o que é necessário para suporte ao microcontrolador. Basta ligar ligá-lo ao computador através de USB e está preparado para começar.

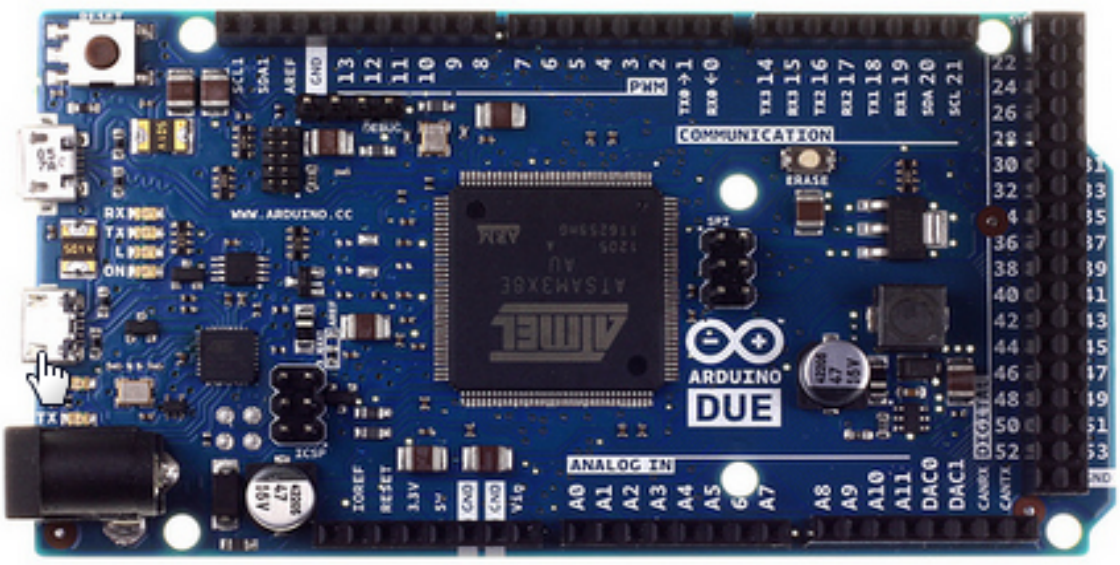


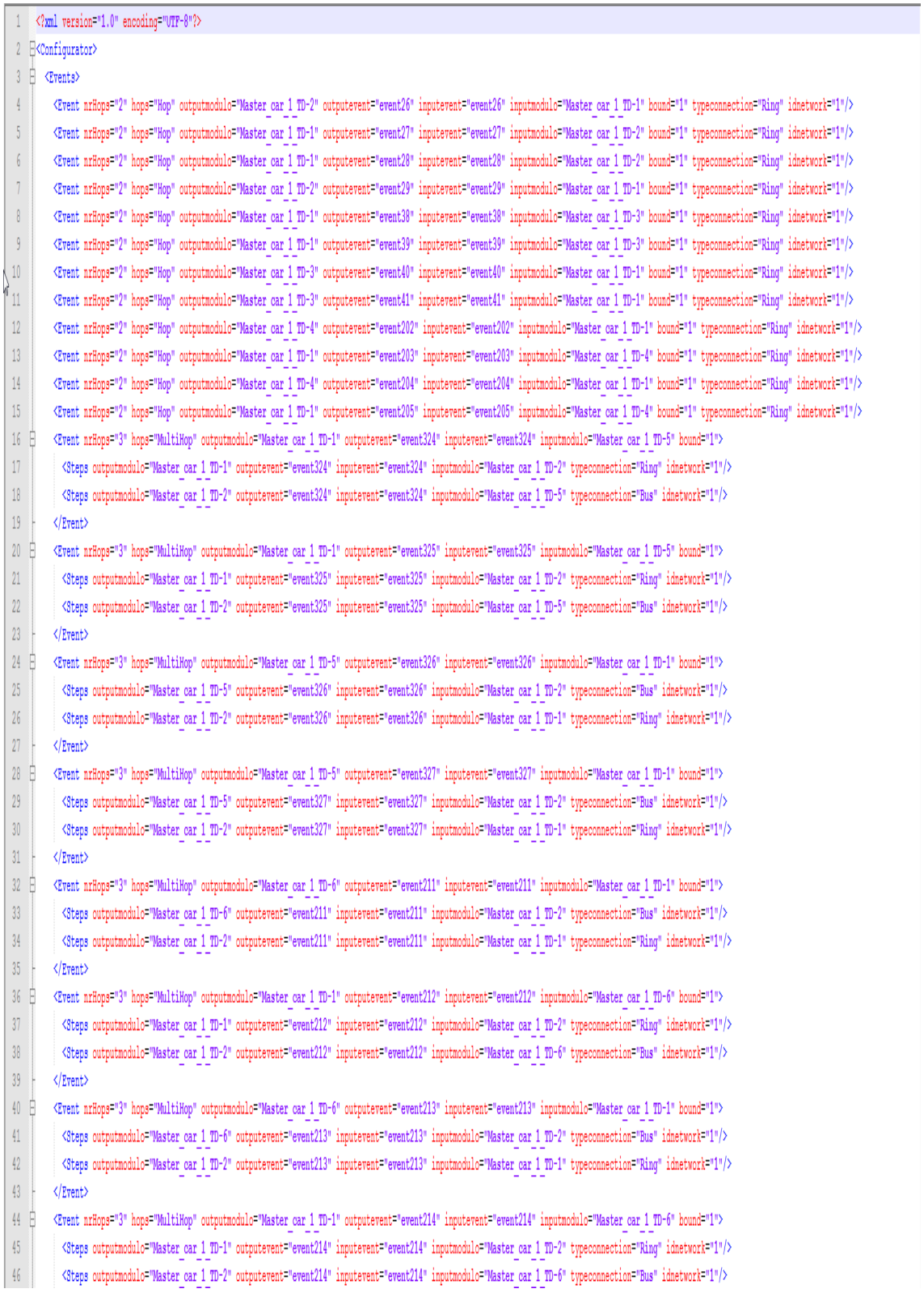
Figura A.11: Arduino Due [1]

Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Outputs Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g

Figura A.12: Características do Arduino Due [1]

Apêndice B

B.1 Apresentação do Ficheiro gerado pela ferramenta



```
<?xml version="1.0" encoding="UTF-8"?>
<Configurator>
  <Events>
    <Event nrHops="2" hops="Hop" outputmodulo="Master_car_1_TD-2" outputevent="event26" inputevent="event26" inputmodulo="Master_car_1_TD-1" bound="1" typeconnection="Ring" idnetwork="1"/>
    <Event nrHops="2" hops="Hop" outputmodulo="Master_car_1_TD-1" outputevent="event27" inputevent="event27" inputmodulo="Master_car_1_TD-2" bound="1" typeconnection="Ring" idnetwork="1"/>
    <Event nrHops="2" hops="Hop" outputmodulo="Master_car_1_TD-1" outputevent="event28" inputevent="event28" inputmodulo="Master_car_1_TD-2" bound="1" typeconnection="Ring" idnetwork="1"/>
    <Event nrHops="2" hops="Hop" outputmodulo="Master_car_1_TD-2" outputevent="event29" inputevent="event29" inputmodulo="Master_car_1_TD-1" bound="1" typeconnection="Ring" idnetwork="1"/>
    <Event nrHops="2" hops="Hop" outputmodulo="Master_car_1_TD-1" outputevent="event38" inputevent="event38" inputmodulo="Master_car_1_TD-3" bound="1" typeconnection="Ring" idnetwork="1"/>
    <Event nrHops="2" hops="Hop" outputmodulo="Master_car_1_TD-1" outputevent="event39" inputevent="event39" inputmodulo="Master_car_1_TD-3" bound="1" typeconnection="Ring" idnetwork="1"/>
    <Event nrHops="2" hops="Hop" outputmodulo="Master_car_1_TD-3" outputevent="event40" inputevent="event40" inputmodulo="Master_car_1_TD-1" bound="1" typeconnection="Ring" idnetwork="1"/>
    <Event nrHops="2" hops="Hop" outputmodulo="Master_car_1_TD-3" outputevent="event41" inputevent="event41" inputmodulo="Master_car_1_TD-1" bound="1" typeconnection="Ring" idnetwork="1"/>
    <Event nrHops="2" hops="Hop" outputmodulo="Master_car_1_TD-4" outputevent="event202" inputevent="event202" inputmodulo="Master_car_1_TD-1" bound="1" typeconnection="Ring" idnetwork="1"/>
    <Event nrHops="2" hops="Hop" outputmodulo="Master_car_1_TD-1" outputevent="event203" inputevent="event203" inputmodulo="Master_car_1_TD-4" bound="1" typeconnection="Ring" idnetwork="1"/>
    <Event nrHops="2" hops="Hop" outputmodulo="Master_car_1_TD-4" outputevent="event204" inputevent="event204" inputmodulo="Master_car_1_TD-1" bound="1" typeconnection="Ring" idnetwork="1"/>
    <Event nrHops="2" hops="Hop" outputmodulo="Master_car_1_TD-1" outputevent="event205" inputevent="event205" inputmodulo="Master_car_1_TD-4" bound="1" typeconnection="Ring" idnetwork="1"/>
    <Event nrHops="3" hops="MultiHop" outputmodulo="Master_car_1_TD-1" outputevent="event324" inputevent="event324" inputmodulo="Master_car_1_TD-5" bound="1">
      <Steps outputmodulo="Master_car_1_TD-1" outputevent="event324" inputevent="event324" inputmodulo="Master_car_1_TD-2" typeconnection="Ring" idnetwork="1"/>
      <Steps outputmodulo="Master_car_1_TD-2" outputevent="event324" inputevent="event324" inputmodulo="Master_car_1_TD-5" typeconnection="Bus" idnetwork="1"/>
    </Event>
    <Event nrHops="3" hops="MultiHop" outputmodulo="Master_car_1_TD-1" outputevent="event325" inputevent="event325" inputmodulo="Master_car_1_TD-5" bound="1">
      <Steps outputmodulo="Master_car_1_TD-1" outputevent="event325" inputevent="event325" inputmodulo="Master_car_1_TD-2" typeconnection="Ring" idnetwork="1"/>
      <Steps outputmodulo="Master_car_1_TD-2" outputevent="event325" inputevent="event325" inputmodulo="Master_car_1_TD-5" typeconnection="Bus" idnetwork="1"/>
    </Event>
    <Event nrHops="3" hops="MultiHop" outputmodulo="Master_car_1_TD-5" outputevent="event326" inputevent="event326" inputmodulo="Master_car_1_TD-1" bound="1">
      <Steps outputmodulo="Master_car_1_TD-5" outputevent="event326" inputevent="event326" inputmodulo="Master_car_1_TD-2" typeconnection="Bus" idnetwork="1"/>
      <Steps outputmodulo="Master_car_1_TD-2" outputevent="event326" inputevent="event326" inputmodulo="Master_car_1_TD-1" typeconnection="Ring" idnetwork="1"/>
    </Event>
    <Event nrHops="3" hops="MultiHop" outputmodulo="Master_car_1_TD-5" outputevent="event327" inputevent="event327" inputmodulo="Master_car_1_TD-1" bound="1">
      <Steps outputmodulo="Master_car_1_TD-5" outputevent="event327" inputevent="event327" inputmodulo="Master_car_1_TD-2" typeconnection="Bus" idnetwork="1"/>
      <Steps outputmodulo="Master_car_1_TD-2" outputevent="event327" inputevent="event327" inputmodulo="Master_car_1_TD-1" typeconnection="Ring" idnetwork="1"/>
    </Event>
    <Event nrHops="3" hops="MultiHop" outputmodulo="Master_car_1_TD-6" outputevent="event211" inputevent="event211" inputmodulo="Master_car_1_TD-1" bound="1">
      <Steps outputmodulo="Master_car_1_TD-6" outputevent="event211" inputevent="event211" inputmodulo="Master_car_1_TD-2" typeconnection="Bus" idnetwork="1"/>
      <Steps outputmodulo="Master_car_1_TD-2" outputevent="event211" inputevent="event211" inputmodulo="Master_car_1_TD-1" typeconnection="Ring" idnetwork="1"/>
    </Event>
    <Event nrHops="3" hops="MultiHop" outputmodulo="Master_car_1_TD-1" outputevent="event212" inputevent="event212" inputmodulo="Master_car_1_TD-6" bound="1">
      <Steps outputmodulo="Master_car_1_TD-1" outputevent="event212" inputevent="event212" inputmodulo="Master_car_1_TD-2" typeconnection="Ring" idnetwork="1"/>
      <Steps outputmodulo="Master_car_1_TD-2" outputevent="event212" inputevent="event212" inputmodulo="Master_car_1_TD-6" typeconnection="Bus" idnetwork="1"/>
    </Event>
    <Event nrHops="3" hops="MultiHop" outputmodulo="Master_car_1_TD-6" outputevent="event213" inputevent="event213" inputmodulo="Master_car_1_TD-1" bound="1">
      <Steps outputmodulo="Master_car_1_TD-6" outputevent="event213" inputevent="event213" inputmodulo="Master_car_1_TD-2" typeconnection="Bus" idnetwork="1"/>
      <Steps outputmodulo="Master_car_1_TD-2" outputevent="event213" inputevent="event213" inputmodulo="Master_car_1_TD-1" typeconnection="Ring" idnetwork="1"/>
    </Event>
    <Event nrHops="3" hops="MultiHop" outputmodulo="Master_car_1_TD-1" outputevent="event214" inputevent="event214" inputmodulo="Master_car_1_TD-6" bound="1">
      <Steps outputmodulo="Master_car_1_TD-1" outputevent="event214" inputevent="event214" inputmodulo="Master_car_1_TD-2" typeconnection="Ring" idnetwork="1"/>
      <Steps outputmodulo="Master_car_1_TD-2" outputevent="event214" inputevent="event214" inputmodulo="Master_car_1_TD-6" typeconnection="Bus" idnetwork="1"/>
```

Figura B.1: Parte 1 do Ficheiro gerado

```
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event214" inputevent="event214" inputmodulo="Master_car_1_TD-6" typeconnection="Bus" idnetwork="1"/>
  </Event>
  <Event nrHops="4" hops="MultiHop" outputmodulo="Master_car_1_TD-1" outputevent="event328" inputevent="event328" inputmodulo="Master_car_1_TD-7" bound="1">
    <Steps outputmodulo="Master_car_1_TD-1" outputevent="event328" inputevent="event328" inputmodulo="Master_car_1_TD-2" typeconnection="Ring" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event328" inputevent="event328" inputmodulo="Master_car_1_TD-6" typeconnection="Bus" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event328" inputevent="event328" inputmodulo="Master_car_1_TD-7" typeconnection="Serial" idnetwork=""/>
  </Event>
  <Event nrHops="4" hops="MultiHop" outputmodulo="Master_car_1_TD-1" outputevent="event329" inputevent="event329" inputmodulo="Master_car_1_TD-7" bound="1">
    <Steps outputmodulo="Master_car_1_TD-1" outputevent="event329" inputevent="event329" inputmodulo="Master_car_1_TD-2" typeconnection="Ring" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event329" inputevent="event329" inputmodulo="Master_car_1_TD-6" typeconnection="Bus" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event329" inputevent="event329" inputmodulo="Master_car_1_TD-7" typeconnection="Serial" idnetwork=""/>
  </Event>
  <Event nrHops="4" hops="MultiHop" outputmodulo="Master_car_1_TD-7" outputevent="event330" inputevent="event330" inputmodulo="Master_car_1_TD-1" bound="1">
    <Steps outputmodulo="Master_car_1_TD-7" outputevent="event330" inputevent="event330" inputmodulo="Master_car_1_TD-6" typeconnection="Serial" idnetwork=""/>
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event330" inputevent="event330" inputmodulo="Master_car_1_TD-2" typeconnection="Bus" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event330" inputevent="event330" inputmodulo="Master_car_1_TD-1" typeconnection="Ring" idnetwork="1"/>
  </Event>
  <Event nrHops="4" hops="MultiHop" outputmodulo="Master_car_1_TD-7" outputevent="event331" inputevent="event331" inputmodulo="Master_car_1_TD-1" bound="1">
    <Steps outputmodulo="Master_car_1_TD-7" outputevent="event331" inputevent="event331" inputmodulo="Master_car_1_TD-6" typeconnection="Serial" idnetwork=""/>
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event331" inputevent="event331" inputmodulo="Master_car_1_TD-2" typeconnection="Bus" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event331" inputevent="event331" inputmodulo="Master_car_1_TD-1" typeconnection="Ring" idnetwork="1"/>
  </Event>
  <Event nrHops="4" hops="MultiHop" outputmodulo="Master_car_1_TD-8" outputevent="event215" inputevent="event215" inputmodulo="Master_car_1_TD-1" bound="1">
    <Steps outputmodulo="Master_car_1_TD-8" outputevent="event215" inputevent="event215" inputmodulo="Master_car_1_TD-6" typeconnection="Serial" idnetwork=""/>
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event215" inputevent="event215" inputmodulo="Master_car_1_TD-2" typeconnection="Bus" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event215" inputevent="event215" inputmodulo="Master_car_1_TD-1" typeconnection="Ring" idnetwork="1"/>
  </Event>
  <Event nrHops="4" hops="MultiHop" outputmodulo="Master_car_1_TD-1" outputevent="event216" inputevent="event216" inputmodulo="Master_car_1_TD-8" bound="1">
    <Steps outputmodulo="Master_car_1_TD-1" outputevent="event216" inputevent="event216" inputmodulo="Master_car_1_TD-2" typeconnection="Ring" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event216" inputevent="event216" inputmodulo="Master_car_1_TD-6" typeconnection="Bus" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event216" inputevent="event216" inputmodulo="Master_car_1_TD-8" typeconnection="Serial" idnetwork=""/>
  </Event>
  <Event nrHops="4" hops="MultiHop" outputmodulo="Master_car_1_TD-8" outputevent="event217" inputevent="event217" inputmodulo="Master_car_1_TD-1" bound="1">
    <Steps outputmodulo="Master_car_1_TD-8" outputevent="event217" inputevent="event217" inputmodulo="Master_car_1_TD-6" typeconnection="Serial" idnetwork=""/>
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event217" inputevent="event217" inputmodulo="Master_car_1_TD-2" typeconnection="Bus" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event217" inputevent="event217" inputmodulo="Master_car_1_TD-1" typeconnection="Ring" idnetwork="1"/>
  </Event>
  <Event nrHops="4" hops="MultiHop" outputmodulo="Master_car_1_TD-1" outputevent="event218" inputevent="event218" inputmodulo="Master_car_1_TD-8" bound="1">
    <Steps outputmodulo="Master_car_1_TD-1" outputevent="event218" inputevent="event218" inputmodulo="Master_car_1_TD-2" typeconnection="Ring" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event218" inputevent="event218" inputmodulo="Master_car_1_TD-6" typeconnection="Bus" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event218" inputevent="event218" inputmodulo="Master_car_1_TD-8" typeconnection="Serial" idnetwork=""/>
  </Event>
  <Event nrHops="4" hops="MultiHop" outputmodulo="Master_car_1_TD-1" outputevent="event332" inputevent="event332" inputmodulo="Master_car_1_TD-9" bound="1">
    <Steps outputmodulo="Master_car_1_TD-1" outputevent="event332" inputevent="event332" inputmodulo="Master_car_1_TD-2" typeconnection="Ring" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event332" inputevent="event332" inputmodulo="Master_car_1_TD-6" typeconnection="Bus" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event332" inputevent="event332" inputmodulo="Master_car_1_TD-9" typeconnection="Serial" idnetwork=""/>
```

Figura B.2: Parte 2 do Ficheiro gerado

```
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event332" inputevent="event332" inputmodulo="Master_car_1_TD-9" typeconnection="Serial" idnetwork=""/>
  </Event>
  <Event nrHops="4" hops="MultiHop" outputmodulo="Master_car_1_TD-1" outputevent="event333" inputevent="event333" inputmodulo="Master_car_1_TD-9" bound="1">
    <Steps outputmodulo="Master_car_1_TD-1" outputevent="event333" inputevent="event333" inputmodulo="Master_car_1_TD-2" typeconnection="Ring" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event333" inputevent="event333" inputmodulo="Master_car_1_TD-6" typeconnection="Bus" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event333" inputevent="event333" inputmodulo="Master_car_1_TD-9" typeconnection="Serial" idnetwork=""/>
  </Event>
  <Event nrHops="4" hops="MultiHop" outputmodulo="Master_car_1_TD-9" outputevent="event334" inputevent="event334" inputmodulo="Master_car_1_TD-1" bound="1">
    <Steps outputmodulo="Master_car_1_TD-9" outputevent="event334" inputevent="event334" inputmodulo="Master_car_1_TD-6" typeconnection="Serial" idnetwork=""/>
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event334" inputevent="event334" inputmodulo="Master_car_1_TD-2" typeconnection="Bus" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event334" inputevent="event334" inputmodulo="Master_car_1_TD-1" typeconnection="Ring" idnetwork="1"/>
  </Event>
  <Event nrHops="4" hops="MultiHop" outputmodulo="Master_car_1_TD-9" outputevent="event335" inputevent="event335" inputmodulo="Master_car_1_TD-1" bound="1">
    <Steps outputmodulo="Master_car_1_TD-9" outputevent="event335" inputevent="event335" inputmodulo="Master_car_1_TD-6" typeconnection="Serial" idnetwork=""/>
    <Steps outputmodulo="Master_car_1_TD-6" outputevent="event335" inputevent="event335" inputmodulo="Master_car_1_TD-2" typeconnection="Bus" idnetwork="1"/>
    <Steps outputmodulo="Master_car_1_TD-2" outputevent="event335" inputevent="event335" inputmodulo="Master_car_1_TD-1" typeconnection="Ring" idnetwork="1"/>
  </Event>
</Events>
<Platforms/>
<Wrappers/>
<Serials>
  <Serial outputmodulo="Master_car_1_TD-6" outputevent="event328" inputevent="event328" inputmodulo="Master_car_1_TD-7" type="Shared" id="0"/>
  <Serial outputmodulo="Master_car_1_TD-6" outputevent="event329" inputevent="event329" inputmodulo="Master_car_1_TD-7" type="Shared" id="0"/>
  <Serial outputmodulo="Master_car_1_TD-7" outputevent="event330" inputevent="event330" inputmodulo="Master_car_1_TD-6" type="Shared" id="0"/>
  <Serial outputmodulo="Master_car_1_TD-7" outputevent="event331" inputevent="event331" inputmodulo="Master_car_1_TD-6" type="Shared" id="0"/>
  <Serial outputmodulo="Master_car_1_TD-8" outputevent="event215" inputevent="event215" inputmodulo="Master_car_1_TD-6" type="Shared" id="4"/>
  <Serial outputmodulo="Master_car_1_TD-6" outputevent="event216" inputevent="event216" inputmodulo="Master_car_1_TD-8" type="Shared" id="4"/>
  <Serial outputmodulo="Master_car_1_TD-8" outputevent="event217" inputevent="event217" inputmodulo="Master_car_1_TD-6" type="Shared" id="4"/>
  <Serial outputmodulo="Master_car_1_TD-6" outputevent="event218" inputevent="event218" inputmodulo="Master_car_1_TD-8" type="Shared" id="4"/>
  <Serial outputmodulo="Master_car_1_TD-6" outputevent="event332" inputevent="event332" inputmodulo="Master_car_1_TD-9" type="Shared" id="8"/>
  <Serial outputmodulo="Master_car_1_TD-6" outputevent="event333" inputevent="event333" inputmodulo="Master_car_1_TD-9" type="Shared" id="8"/>
  <Serial outputmodulo="Master_car_1_TD-9" outputevent="event334" inputevent="event334" inputmodulo="Master_car_1_TD-6" type="Shared" id="8"/>
  <Serial outputmodulo="Master_car_1_TD-9" outputevent="event335" inputevent="event335" inputmodulo="Master_car_1_TD-6" type="Shared" id="8"/>
  <Rate serialid="0" baudrate="2400" stopbit="1"/>
  <Rate serialid="4" baudrate="2400" stopbit="1"/>
  <Rate serialid="8" baudrate="2400" stopbit="1"/>
</Serials>
<Buses>
  <Networkbus id="1" busnetworktype="Ethernet">
    <Bus modulo="Master_car_1_TD-2" position="Pos_1"/>
    <Bus modulo="Master_car_1_TD-5" position="Pos_2"/>
    <Bus modulo="Master_car_1_TD-6" position="Pos_3"/>
  </Networkbus>
</Buses>
<Rings>
  <Networkring id="1" ringnetworktype="Serial">
```

Figura B.3: Parte 3 do Ficheiro gerado

```
136     <Networkring id="1" ringnetworktype="Serial">
137       <Ring modulo="Master_car_1_TD-2" position="Pos_2"/>
138       <Ring modulo="Master_car_1_TD-1" position="Pos_1"/>
139       <Ring modulo="Master_car_1_TD-3" position="Pos_3"/>
140       <Ring modulo="Master_car_1_TD-4" position="Pos_4"/>
141     </Networkring>
142   </Rings>
143   <DoubleRings/>
144   <Matrixs/>
145   <Toroidals/>
146 </Configurator>
```

Figura B.4: Parte 4 do Ficheiro gerado